Attacking and Defending Code-based Cryptosystems

Towards secure and efficient cryptographic applications based on error-correcting codes

Vom Fachbereich Informatik der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades
Doctor rerum naturalium (Dr. rer. nat.)
von

Dipl.-Math. Robert Niebuhr

geboren in Offenbach am Main.

Referenten:	Prof. Dr. Johannes Buchmann Prof. Dr. Paulo Barreto Prof. Dr. Pierre-Louis Cayrel
Tag der Einreichung:	21.5.2012
Tag der mündlichen Prüfung:	28.6.2012
Hochschulkennziffer:	D 17

Darmstadt 2012

Wissenschaftlicher Werdegang

Juli 2008 – heute

Promotionsstudium am Lehrstuhl von Prof. Dr. Johannes Buchmann, Fachbereich Informatik, Fachgebiet Theoretische Informatik — Kryptographie und Computeralgebra, Technische Universität Darmstadt.

Juli 2003 – Juni 2004

Auslandsstudium an der University of New South Wales, Sydney, Australien.

April 2001 – März 2006

Studium der Mathematik mit Nebenfach Informatik an der Technischen Universität Darmstadt.

List of Publications

[1] P. S. L. M. Barreto, P.-L. Cayrel, R. Misoczki, and R. Niebuhr. Quasi-dyadic CFS signatures. In *Inscrypt 2010*, volume 6584 of *Lecture Notes in Computer Science*. Springer, 2010. Cited on pages 62, 79, 83, 101, and 114.

[2] P.-L. Cayrel, M. ElYousfi, G. Hoffman, M. Meziani, and R. Niebuhr. Recent progress in code-based cryptography. In *ISA 2011*, volume CCIS 200 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 2011.

[3] R. Niebuhr. Critical attacks in code-based cryptography. In *WEWoRC 2011*, 2011. Cited on page 34.

[4] R. Niebuhr. Statistical decoding of codes over $\mathbb{F}_q$. 2011. Accepted at PQCrypto 2011.

[5] R. Niebuhr and P.-L. Cayrel. Broadcast attacks against code-based encryption schemes. Lecture Notes in Computer Science. Springer, 2011. Accepted for WEWoRC 2011 post-proceedings. Cited on page 35.

[6] R. Niebuhr, P.-L. Cayrel, and J. Buchmann. Improving the efficiency of Generalized Birthday Attacks against certain structured cryptosystems. In *WCC 2011*, pages 163–172. Springer, Apr 2011. Cited on pages 58 and 79.

[7] R. Niebuhr, P.-L. Cayrel, S. Bulygin, and J. Buchmann. Attacking code/lattice-based cryptosystems using Partial Knowledge. In *Inscrypt 2010*. Science Press of China, 2010.

[8] R. Niebuhr, P.-L. Cayrel, S. Bulygin, and J. Buchmann. On lower bounds for Information Set Decoding over $\mathbb{F}_q$. In *SCC 2010, RHUL, London, UK*, pages 143–157, 2010. Cited on pages 65, 85, 92, 105, 106, and 109.

[9] R. Niebuhr, P.-L. Cayrel, S. Bulygin, and J. Buchmann. On lower bounds for Information Set Decoding over $\mathbb{F}_q$ and on the effect of Partial Knowledge. Submitted to Math in CS (SCC 2010), 2011. Cited on pages 92 and 105.

[10] R. Niebuhr, M. Meziani, S. Bulygin, and J. Buchmann. Selecting Parameters for Secure McEliece-based Cryptosystems. *International Journal of Information Security*, 2011. Cited on pages 37 and 109.

Acknowledgements

First and foremost, I thank Johannes Buchmann for giving me the opportunity to accomplish my dissertation in his group and for his continuous constructive criticism. He gave me the freedom to pursue my research interests and allowed me to travel the world to be part of the international research community.

I am grateful to Pierre-Louis Cayrel for supervising my thesis, for numerous contributions and many interesting discussions. He challenged my work whenever necessary.
Further, I thank Paulo Barreto for the fruitful joint work and for agreeing to be my co-referee.

For interesting and beneficial collaboration I thank Stanislav Bulygin, Mohamed El-Yousfi, Gerhard Hoffman, Mohammed Meziani, Rafael Misoczki, Raphael Overbeck, and Falko Strenzke. I am especially grateful to Nicolas Sendrier for hosting us in Paris and for many hours of helpful and instructive discussions.

Furthermore, I thank my CDC and CASED colleagues for creating a pleasant office atmosphere. Without Marita Skrobić and Roswitha Jäger-Beck, many things would not run as smoothly as they do.

My special thanks go to Hans, Ilona, Laura, and Verena. Their mixture of encouragement, constructive criticism, questions, and support helped me greatly to accomplish this undertaking.

Zusammenfassung

Gegenwärtig werden kryptographische Anwendungen in fast allen Bereichen des Lebens genutzt, einschließlich Wirtschaft, Gesundheitswesen, Militär und Unterhaltung. Ohne sie würde sich unser Leben in unvorhersehbarer Weise verändern. Seit der Veröffentlichung von Shor's Algorithmus in 1994 ist allerdings bekannt, dass kryptographische Anwendungen, die auf den Problemen des Faktorisierens oder diskreten Logarithmus basieren, von Quantencomputer-Angriffen bedroht sind. Dazu gehören fast alle heutigen Anwendung.
Es wird davon ausgegangen, dass code-basierte Kryptographie sicher ist gegen Angriffe von Quantencomputern. Darüber hinaus hat sie einige weitere Vorteile. Erstens weist sie gute Sicherheits-Eigenschaften auf: Beispielsweise werden binäre Goppacodes als eine sichere Wahl für code-basierte Anwendungen angesehen, und das McEliece-Verschlüsselungsverfahren wurde seit seiner Veröffentlichung vor über 30 Jahren nicht gebrochen (es war lediglich eine Anpassungen der Parameter erforderlich).
Zweitens können code-basierte Anwendungen vergleichsweise einfach auf Geräten mit geringer Rechenkapazität und ohne kryptographischem Coprozessor laufen, da sie nur lineare Algebra nutzen, anstelle von z.B. Fließkommaberechnungen.
Schließlich kann die Komplexität von Angriffen gegen code-basierte Anwendungen meist präzise in der erwarteten Anzahl von Operationen berechnet werden, ohne die asymptotische $\mathcal{O}$-Notation nutzen zu müssen. Dies erlaubt eine genaue Berechnung des Sicherheitslevels dieser Anwendungen. Der hauptsächliche Nachteil der meisten code-basierten Anwendungen ist die beträchtliche Schlüsselgröße von mehreren Kilobyte bis Megabyte.

Diese Arbeit leistet einen Beitrag zu zwei wichtigen Aspekten der Entwicklung kryptographischer Anwendungen. Der erste betrifft die Verwendung sicherer Primitive. Während Verfahren wie McEliece und Niederreiter seit langer Zeit analysiert werden und heute als sicher gelten, werden kontinuierlich Varianten und neue Verfahren entwickelt. Die Zielsetzung ist typischerweise eine Verringerung der Schlüsselgröße oder die Präsentation neuer Eigenschaften und anderer Verbesserungen (wie z.B. höhere Effizienz). Zur Bestimmung der Sicherheit dieser neuen Verfahren, müssen diese gegen alle relevanten Angriffe getestet werden. Unser Beitrag besteht in der Entwicklung eines neuen Angriffs, des Broadcast-Angriffs, sowie der Weiterentwicklung und Generalisierung existierender Angriffe.
Der zweite Aspekt betrifft die Wahl sicherer Parameter, welche für eine praktische Anwendung der Primitive erforderlich ist. Die hierfür zu berücksichtigen Einschränkungen sind in erster Linie die Gewährleistung eines ausreichenden Sicherheitslevels; weitere betreffen verschiedene Aspekte von Effizienz, z.B. Ver- und Entschlüsselungszeit, erforderliche Bandbreite oder Speichergröße usw. Unser Beitrag ist die Bestimmung optimaler Parameter für das McEliece-Verschlüsselungsverfahren sowie das QD-CFS Signaturschema durch Anwendung des Systems von Lenstra und Verheul.

Abstract

Today, cryptographic applications are used in nearly all areas of our lives, including the economy, health, military, and entertainment. Without them, society would change in ways we can hardly imagine. Since the publication of Shor's algorithm in 1994, however, we know that those cryptographic applications based on the problems of factoring and discrete logarithm are threatened by quantum computer attacks. Most current applications belong to this category.
Code-based cryptography is conjectured to be secure against quantum computer attacks, and it has several other advantages. Firstly, it exhibits advanced security properties: for example, binary Goppa codes are considered a secure choice for many schemes, and the McEliece encryption scheme has not been broken in over 30 years since its publication (merely an adjustment of the parameters was necessary).
Secondly, since code-based cryptosystems are based on linear algebra instead of, for example, arithmetic using floating-point numbers, they are usually very fast and can be implemented on devices with a low computing power and without cryptographic co-processor.
Finally, the complexity of attacks against code-based schemes can usually be estimated accurately in the expected number of binary operations instead of relying on the asymptotic $\mathcal{O}$-notation. This allows a precise computation of the security level a scheme provides.
The major drawback of most code-based schemes is their large key size.

This thesis contributes to two important aspects in the development of cryptographic applications. The first concerns the use of secure cryptographic primitives. While schemes like the McEliece and Niederreiter encryption schemes have been studied for a long time and are considered secure, new schemes or variants of existing ones are constantly being developed. The objectives are to decrease the key size, create schemes with new properties, or to introduce other improvements. In order to assess the security of these schemes, they have to be subjected to all relevant attacks. We contribute to this aspect by introducing a new type of attack, a broadcast attack, and by improving and generalizing existing attacks.
Secondly, once a secure primitive has been found, appropriate code parameters have to be selected. This choice needs to reflect several constraints. Most importantly, the parameters have to provide a sufficient security level. Other constraints concern different aspects of efficiency, e.g. encryption or decryption time, required bandwidth, memory size etc. Our contribution is the selection of optimal parameters for the McEliece cryptosystem and the QD-CFS signature scheme by applying Lenstra and Verheul's framework.

Contents

1. Introduction

Throughout history, cryptography has played an important role for society. For instance, Julius Caesar sent encrypted messages to his troops, and breaking the famous Enigma ciphers in World War 2 was a critical success in the Allied war effort. While historically most applications were military, today cryptography is used in many other areas as well: Encryption schemes allow private and secure communication; they protect digital rights, e.g. the content of DVDs or pay-TV streams; and they secure sensitive data on all levels, be it national secrets or personal medical records. Digital signatures certify the source of documents and ensure that they are unaltered; this does not only include the common example of signed emails, but in fact, digital signatures are the basis of public-key infrastructure, creating a hierarchy of trust and thereby enabling applications such as electronic banking or credit card transactions via the internet. Identification and authentication schemes ensure that the communicating parties really are who they claim. Therefore, they are an important component of many of the applications mentioned above. These tools are thus essential for our modern economy, since global trade and the increasing number of electronic financial transactions would hardly be possible, if at all.
In order to allow communication between two parties without the need to meet or exchange keys via some other channel, public-key cryptography is required. In contrast to secret-key cryptography, public-key schemes do not require the initial secret exchange of some key. Every user has a public and a private/secret key. The public key is used, for example, to encrypt messages addressed to this user and to verify his signatures, while the decryption of messages as well as the generation of signatures require the secret key.
All public-key schemes are based on a (usually mathematical) hard problem. This means that the scheme can be attacked by solving an instance of this problem, and in some cases the (stronger) other direction holds as well. Most cryptographic schemes used today are based on the problem of factoring large integers or on the problem of computing discrete logarithms, e.g. the RSA and ElGamal encryption schemes. However, in 1994 P. Shor published a quantum algorithm allowing to solve these two problems in polynomial time, showing that large enough quantum computers can break these "classical" cryptosystems. While existing quantum computers use only a small number of quantum bits, or qubits, and while the technical difficulties to employ more qubits are very high, many researchers expect that large quantum computers will be built in the next few decades.
It is, therefore, crucial to develop cryptosystems that are resistant to quantum computer attacks. Error-correcting codes, originally developed for communication over noisy channels, have been applied in cryptography for at least three decades, ever since R. J. McEliece published his encryption scheme in 1978. Together with cryptography based on lattices, multivariate equations, and hash functions, code-based cryptography is conjectured to be secure against quantum computer attacks. Consequently, it is a promising candidate for post-quantum cryptography, which refers to those cryptographic schemes that remain se-

cure after the development of large quantum computers.
The advancement of code-based cryptography takes place in four areas: Firstly, the development of secure cryptosystems. Progress in this area requires constant improvement of attacks to assess the security of a scheme as precisely as possible. Secondly, the choice of secure and efficient parameters for these schemes. This selection has to take into account the specific cryptosystem, the time for which protection is required, and assumptions about future hardware and software developments. The third area is the reduction of the key size which is usually very large for code-based cryptosystems. Research towards this goal is linked with the first two areas since many proposals for key size reduction have implications on the security of the respective scheme. Finally, the development of schemes with additional features, for instance blind or ring signatures.

1.1. Summary of results

The goal of this thesis is to improve the understanding of the theoretical security and to advance the applicability of code-based cryptography. We focus on the first two of the four areas mentioned above: the development and improvement of attacks against code-based cryptosystems (Chapters 3 and 4) and the choice of secure and efficient parameters (Chapter 5). More specifically, this work contributes the following results.

Chapter 3: Critical attacks Our first contribution consists of the results of a comprehensive analysis of broadcast attacks against the Niederreiter and the HyMES cryptosystems. A broadcast scenario is given if a sender broadcasts a message m to several users, encrypted with the respective user's public key. A typical example for this scenario is the distribution of standardized emails to many recipients. We show that both schemes are vulnerable to this kind of attack if they are not protected by a layer of semantic conversion. Our attack is related to the one by Plantard and Susilo who studied broadcast attacks against lattice-based schemes. Here, we extend their approach by the following aspects: we prove an explicit bound for the expected number of recipients that is required to run our attack; we cover the case where too few messages are intercepted by an attacker and analyze the situation where the broadcasted messages are not identical but similar to each other; finally, we discuss in detail the use of semantic conversions to protect against broadcast attacks. An implementation of our attack can be found in the appendix of this work (see page 135).

Chapter 4: Non-critical attacks The second group of contributions consists of improvements of non-critical attacks. The first is a generalization of attacks based on information set decoding from binary to larger fields $\mathbb{F}_q$. We propose and prove lower bounds for the complexity of these algorithms, and we show that the field structure can be used to improve the efficiency compared with a straightforward generalization. Secondly, we analyze the effect of partial knowledge on the efficiency of attacks. We define two types of partial knowledge an attacker might be able to obtain, describe techniques that exploit this knowledge, and prove lower bounds for the complexity of attacks employing these

techniques. Our third result is the improvement of attacks based on the generalized birthday algorithm for the case where the target matrix is of a certain structure. We conclude this section with an analysis of three "cross-area" attacks: two lattice-based attacks used against code-based systems and one code-based attack employed against lattice-based systems.

Chapter 5: Parameter selection Our final group of results focus on the major practical aspect of code-based cryptography, the selection of secure parameters. In contrast to some other areas of cryptography where the complexity of attacks is known only asymptotically (and expressed in $\mathcal{O}$-notation), the complexity of many attacks against code-based cryptosystems can be expressed in the expected number of binary operations. This facilitates a detailed analysis of secure and efficient parameters.
We begin with an application of Lenstra and Verheul's framework to the McEliece cryptosystem. Using this framework and a set of assumptions about future hardware and software developments, we derive parameters that are expected to be secure at least until a given year and minimize the corresponding public key size. Next, we focus on the selection of optimal parameters for the QD-CFS signature scheme. Our analysis is also based on Lenstra and Verheul's framework, and it takes into account the various constraints which are due to structural attacks and the use of quasi-dyadic codes. Finally, we show that the selection of parameters allows to trade off signature generation time versus public key size.

1.2. Related Work

This section provides a context for this thesis. We reference the major publications from the four research areas listed above.
The first notable code-based encryption scheme was published by R. J. McEliece [55] in 1978. The Niederreiter cryptosystem [70], which can be seen as a dual to McEliece, was presented in 1986. Both schemes remain secure up to date (when used with Goppa codes and appropriate parameters). Other important code-based schemes are the CFS signature scheme [70], the FSB hash function [5] and the SYND stream cipher [39].
Many attacks have been developed against these and other cryptosystems. Two important types of generic attacks are information-set decoding and the generalized birthday algorithm. Initially proposed by Prange [79], the former was improved and refined many times (see Sections 2.5.2 and 4). The latter was developed by Wagner [97] and was generalized by Minder and Sinclair [60] (see Sections 2.5.2 and 4.3). In addition to that, there have been several structural attacks against specific cryptosystems. Examples include the Sidelnokiv-Shestakov attack [87] in 1992 and the algebraic cryptanalysis of McEliece variants with compact keys by Faugère et al. [33].
While many publications of new schemes included sample parameters, we are not aware of a detailed, comprehensive analysis of the parameter selection that compares to ours.
Key size reduction was not a major research focus until it received increasing attention in recent years. This development is in great measure due to the emergence and growing use of small devices, e.g. mobile phones and sensors, which often contain very limited memory.

Many proposals are based on the idea to use codes with additional structure which allows a more compact representation, for instance low-density-parity-check (LDPC), quasi-cyclic, and quasi-dyadic codes [40, 10, 61].
The development of cryptosystems with additional properties is another recent trend. Code-based ring signatures [98], identity-based identification and signature schemes [25], and threshold ring signatures [56, 30] are prominent examples.
In summary, code-based cryptography has received increasing attention in recent years, but there is much room for further improvements towards secure and efficient cryptographic applications.

2. Preliminaries, Definitions, and Notation

In this section, we review notions and definitions from coding theory, code-based cryptography, and cryptanalysis. We aim to make this thesis as self-contained as possible, but we will confine ourselves to the most important parts and provide references for more details. A collection of symbols and abbreviations used in this thesis can be found on page 125.

2.1. Introduction

Coding theory is the study of error-correcting codes and their properties. The study of these codes was inspired by Shannon's article "A mathematical theory of communication" [85]. They were initially developed to correct errors due to data transmission over noisy channels, and there is a wide range of applications including CDs and DVDs, hard disks, mobile phones, and satellite communication. Codes are also used for data compression, and they are the foundation of code-based cryptography.
The underlying idea is to add redundancy to the data which allows the correction of a certain number and type of errors. Specifically, the data or message is mapped injectively into a certain set which is called a code, and its elements are codewords. Encoding a message is the application of this map; depending on the context, decoding refers either to the process of removing the errors from a codeword, or to the map from codewords potentially containing errors to the set of messages. We will use the former definition in this thesis.
In code-based cryptography, the sender of a message intentionally adds errors to the codeword in order to make decoding — and thus decryption — difficult. The receiver of the message is able to decode it by using some secret knowledge (usually about the code structure), but this is infeasible for an attacker who does not have the secret knowledge.

2.2. Mathematical notation

In this thesis, we denote row vectors and numbers by lower case, and matrices and sets by upper case roman letters, unless the notation commonly used in the literature is different. We denote by $\mathcal{I}_k$ the set $\{1, 2, \ldots, k\}$. The elements that vectors and matrices consist of are also called entries or bits, and they are denoted by subscripts, e.g. $x = (x_1, \ldots, x_n)$ and $H = (H_{ij})_{i,j \in \mathcal{I}_n}$. For any non-zero vector $v \in \mathbb{F}_q^n \backslash \{0\}$, $\mathrm{le}(v) \in \mathbb{F}_q$ denotes the leading element of v, i.e. its first non-zero element. As usual, addition of vectors and matrices is defined on the entries. In general, we denote multiplication by $\cdot$, but we will omit it where this improves the readability. Whenever the summands or factors are elements of a finite field $\mathbb{F}_q$ and unless stated otherwise, the operation is performed over $\mathbb{F}_q$. For matrices A,

B, and C, we write $A = (B|C)$ and $A' = \langle B, C\rangle$ to denote the horizontal and vertical concatenation of B and C, respectively, and similarly for vectors. For a matrix A and $J \subseteq \mathcal{I}_n$, $A_{.J}$ denotes the submatrix of A consisting of those columns indexed by the set J.

2.3. Coding theory

In the following, we recall selected notions from coding theory. We refer to the book by MacWilliams and Sloane [53] for more details.

2.3.1. General concepts

Definition 2.3.1 (Error-correcting code)**.** A (linear) error-correcting code $\mathcal{C}$ of length n and dimension k, defined over a finite field $\mathbb{F}_q$, is a k-dimensional vector subspace of the n-dimensional vector space $\mathbb{F}_q^n$. A code with these properties is called an $[n, k]$ code. For $q = 2$, $\mathcal{C}$ is called a binary code, otherwise it is called a q-ary code. The co-dimension r of a code is defined as $r = n - k$, and the code rate R is the ratio of dimension to length, $R = k/n$.

Definition 2.3.2 (Permutation-equivalent codes)**.** Two linear codes $\mathcal{C}$ and $\mathcal{C}'$ are called permutation-equivalent if there exists a permutation σ on $\mathcal{I}_n$ such that

$$\mathcal{C}' = \sigma(\mathcal{C}) := \{(x_{\sigma^{-1}(i)})_{i\in\mathcal{I}_n} : (x_i)_{i\in\mathcal{I}_n} \in \mathcal{C}\}.$$

Definition 2.3.3 (Hamming weight, Hamming distance)**.** The (Hamming) weight $\mathrm{wt}(v)$ of any vector v is defined as the number of non-zero entries of v. The (Hamming) distance of two vectors $\mathrm{d}(x, y)$ is defined as $\mathrm{wt}(x - y)$.

In order to decode words in a set $D \subseteq \mathbb{F}_q^n$ with respect to an $[n, k]$ code $\mathcal{C}$, every word $c \in D$ needs to be associated with a codeword which can then be mapped uniquely to the corresponding message. There are different ways to do this, but the most common one is to associate a given word with that codeword which has minimum distance.

Definition 2.3.4 (Decoding algorithm)**.** Let $\mathcal{C}$ be an $[n, k]$ code over $\mathbb{F}_q$ and $D \subseteq \mathbb{F}_q^n$. An algorithm $\mathcal{D}_\mathcal{C}$ that takes a word $c \in D$ and a natural number t as input is called a t-error-correcting algorithm for $\mathcal{C}$ iff

$$\forall c \in D : \mathcal{D}_\mathcal{C}(c) = \{x \in \mathcal{C} : \mathrm{d}(x, c) \leq t\}.$$

If $D = \mathbb{F}_q^n$, $\mathcal{D}_\mathcal{C}$ allows *complete decoding*.

Two important properties of a code are its minimum distance and its error-correcting capability.

Definition 2.3.5 (Minimum distance, error-correcting capability)**.** The minimum distance d of a code $\mathcal{C}$ is the minimum distance of any two different codewords, or equivalently, the minimum weight of all non-zero codewords:

$$d = \min_{\substack{x,y\in\mathcal{C}\\ x\neq y}} \mathrm{d}(x, y) = \min_{\substack{x\in\mathcal{C}\\ x\neq 0}} \mathrm{wt}(x).$$

A code of length n, dimension k and minimum distance d is denoted an $[n, k, d]$ code. For $c \in \mathbb{F}_q^n$ and any $t \leq \lfloor (d-1)/2 \rfloor$, there can be at most one codeword x with $\mathrm{d}(x, c) \leq t$. Therefore, c can be decoded uniquely to x, and $t = \lfloor (d-1)/2 \rfloor$ is called the error-correcting capability of $\mathcal{C}$. A code with these properties is denoted an (n, k, t) code.

Since a code is a vector subspace, it can be defined by a basis and by its dual space.

Definition 2.3.6 (Generator and parity check matrices, syndrome)**.** Let $\mathcal{C}$ be an $[n, k]$ linear code over $\mathbb{F}_q$. A $k \times n$ matrix G of full rank and defined over $\mathbb{F}_q$ is a generator matrix of $\mathcal{C}$ if the rows of G form a basis of $\mathcal{C}$. We say that G generates $\mathcal{C}$, and we have

$$\mathcal{C} = \{mG : m \in \mathbb{F}_q^k\}.$$

A parity check matrix of $\mathcal{C}$ is an $r \times n$ matrix of full rank and defined over $\mathbb{F}_q$, where

$$\mathcal{C} = \{x \in \mathbb{F}_q^n : Hx^{\mathrm{T}} = 0\}.$$

H is a generator matrix for the dual space $\mathcal{C}^{\perp}$. For a given parity check matrix H and any vector e, we call s the syndrome of e if $s^{\mathrm{T}} = He^{\mathrm{T}}$.

Two generator matrices generate permutation-equivalent codes if one is obtained from the other by a linear transformation and a permutation. Therefore, up to permutation we can write any generator matrix G in *systematic form* $G = [I_k|R]$, where I_k is the identity matrix of size k, which allows a more compact representation. If $\mathcal{C}$ is generated by $G = [I_k|R]$, then a parity check matrix for $\mathcal{C}$ is $H = [-R^{\mathrm{T}}|I_{n-k}]$. Up to permutation, it is also possible to have the identity submatrix on the left hand side of H.

In 1952 and 1957, Gilbert [42] and Varshamov [96] independently developed bounds on the maximum size of a code. While the methods to prove these bounds were different, the bounds are asymptotically equal. Based on these bounds, Barg [7, Section 1.2] proposed the related Gilbert-Varshamov distance.

Definition 2.3.7 (Gilbert-Varshamov (GV) bound, GV distance)**.** Let $A_q(n, d)$ denote the maximum size of an $[n, k, d]$ code over $\mathbb{F}_q$, i.e. the maximum number of codewords. The GV bound states that

$$A_q(n, d) \geq \frac{q^n}{\sum_{j=0}^{d-1} \binom{n}{j}(q-1)^j}.$$

For $\mathcal{C}$ an $[n, k]$ code over $\mathbb{F}_q$, the GV distance is defined as the maximum integer d such that

$$\sum_{j=0}^{d-1} \binom{n}{j}(q-1)^j \leq q^{n-k}.$$

2.3.2. Goppa codes

This section covers special types of codes that will be used in different parts of this thesis. We start with the definition of Goppa codes. These codes have been used in cryptography for more than three decades, ever since McEliece presented his encryption scheme in 1978 [55].

A useful method for constructing codes is the subfield subcode construction (see [90] for details on the construction and on properties of subfield subcodes).

Definition 2.3.8 (Subfield subcodes)**.** Let $\mathcal{C}$ be a code defined over an extention field $\mathbb{F}_{q^m}$ of $\mathbb{F}_q$. A subfield subcode $\mathcal{C}'$ of $\mathcal{C}$ over $\mathbb{F}_q$ is the restriction of $\mathcal{C}$ to $\mathbb{F}_q$:

$$\mathcal{C}' = \mathcal{C}_{|\mathbb{F}_q} = \mathcal{C} \cap \mathbb{F}_q^n.$$

Definition 2.3.9 (Vandermonde matrix)**.** Given a set $L = (L_1, \ldots, L_n) \in \mathbb{F}_q^n$ and an integer $t > 0$, the Vandermonde matrix $\mathrm{vdm}(t, L)$ is the $t \times n$ matrix V with elements $V_{ij} = L_j^{i-1}$.

Definition 2.3.10 (Generalized Reed-Solomon (GRS) and alternant codes)**.** Let $L = (L_1, \ldots, L_n) \in \mathbb{F}_q^n$ be a sequence of distinct elements and $D = (D_1, \ldots, D_n) \in \mathbb{F}_q^n$ be a sequence of non-zero elements. The GRS code $\mathrm{GRS}_t(L, D)$ is the (n, k, t) code defined by the parity-check matrix

$$H = \mathrm{vdm}(t-1, L) \cdot \mathrm{diag}(D).$$

An alternant code is a subfield subcode of a GRS code.

Definition 2.3.11 (Goppa codes)**.** Let q be a prime power, $\mathbb{F}_{q^m}$ an extention field of $\mathbb{F}_q$ of degree $m \geq 1$, a sequence $L = (L_1, \ldots, L_n) \in \mathbb{F}_{q^m}^n$ of distinct elements, and a polynomial $g(x) \in \mathbb{F}_{q^m}[x]$ of degree t such that $g(L_i) \neq 0$ for $1 \leq i \leq n$. The Goppa code $\Gamma(L, g)$ over $\mathbb{F}_q$ is the alternant code over $\mathbb{F}_q$ corresponding to $\mathrm{GRS}_t(L, D)$, where $D = (g(L_0)^{-1}, \ldots, g(L_n)^{-1})$.

2.3.3. QC alternant codes, QD Goppa codes

Due to the main drawback of many code-based cryptosystems, the large public key size, there have been many proposals on how to reduce those key sizes. One way to do this is to use codes with additional structure. These allow highly structured generator and parity check matrices which can be stored more efficiently. Two examples that play a role in this thesis are quasi-cyclic (QC) alternant [10] and quasi-dyadic (QD) Goppa [61] codes. We give a brief description here and refer to the original papers for more details.

Definition 2.3.12 (Circular/cyclic matrices)**.** A circulant or cyclic matrix M is a square matrix where each row is a cyclic shift of the previous

$$M = \begin{pmatrix} m_1 & m_2 & \cdots & m_n \\ m_n & m_1 & \cdots & m_{n-1} \\ \vdots & \ddots & \ddots & \vdots \\ m_2 & \cdots & m_n & m_1 \end{pmatrix}.$$

Definition 2.3.13 (Dyadic matrices)**.** Let $h = (h_0, h_1, \ldots, h_{n-1}) \in \mathbb{F}_q^n$ and $n = 2^a$ for some integer a. A dyadic matrix D is an $n \times n$ matrix over $\mathbb{F}_q$ with entries $D_{ij} = h_{i \oplus j}$, where $\oplus$ refers to bitwise XOR on the binary representations of the indices. The vector h is called the signature of D.

Definition 2.3.14 (QC and QD matrices and codes)**.** A QC (QD) matrix is a $k_0 l \times n_0 l$ block matrix where each block of size $l \times l$ is cyclic (dyadic). A QC (QD) code is one that admits a QC (QD) generator matrix.

Cyclic and dyadic matrices are completely determined by their first row, and QC and QD matrices by the first rows of the $k_0 n_0$ blocks. This allows a much more efficient representation.

In [10], Berger et al. proposed to use QC alternant codes for the McEliece cryptosystem (see Section 2.4.2) and presented an algorithm to construct these codes. Misoczki and Barreto [61] focused on a different class of codes and proposed QD Goppa codes. In order to do that, they first had to show that there are a sufficient number of Goppa codes that admit a QD generator matrix, and they also gave an algorithm to construct these codes.

2.4. Code-based Cryptography

Code-based cryptography exists since more than three decades but it has never gained much public acceptance since the schemes usually have a large public key size. However, it is a candidate for "post-quantum cryptography" as it is considered immune to quantum computer attacks. In contrast, once large enough quantum computers exist, Shor's algorithm [86], published in 1997, allows to break all cryptographic schemes which are based on the integer factoring or the discrete logarithm problem.
In this section, we recall some hard problems from code-based cryptography and describe several cryptographic schemes which will be used in this thesis.

2.4.1. Hard Problems

Most code-based cryptographic schemes are based on the difficulty of one or more of the following hard problems (or a variant of these): The general decoding problem, the syndrome decoding problem, and the code distinguishing problem.

Problem 2.4.1 (General decoding problem)**.** *Given an $[n, k]$ code $\mathcal{C}$ over $\mathbb{F}_q$, an integer t_0 and a vector $c \in \mathbb{F}_q^n$, find a codeword $x \in \mathcal{C}$ with $d(x, c) \leq t_0$ or return $\{\}$ iff no such codeword exists.*

Problem 2.4.2 (Syndrome Decoding (SD) problem)**.** *Given a matrix H and a vector s, both over $\mathbb{F}_q$, and a non-negative integer t_0; find a vector $x \in \mathbb{F}_q^n$ of (Hamming) weight t_0 such that $Hx^T = s^T$ or return $\{\}$ iff no such vector exists.*

Depending on the parameters n, k, and t_0, the expected number of solutions of these decoding problems can be different. For example, in the context of an encryption scheme there is only one solution expected (otherwise decryption would not be unique), while the

problem of finding a collision in a hash function usually has a large number of possible solutions.

Proposition 2.4.3. *Both, the general decoding and the SD problem are equivalent to the problem of finding codewords of low weight in the same or a slightly longer code.*

Proof. For both problems, let $\mathcal{C}$ be the given $[n, k]$ code, generated by a parity check matrix H.
In case of the general decoding problem, let $\mathcal{C}'$ be the code obtained by adding the vector c to $\mathcal{C}$, i.e. the vector space spanned by $\mathcal{C}$ and c. Since $x \in \mathcal{C}$, the new code contains the difference $e = c - x \in \mathcal{C}'$, and $\mathrm{wt}(e) = t_0$. Therefore, the general decoding problem can be solved by finding a vector e of weight not exceeding t_0 in the code $\mathcal{C}'$.
To convert the SD problem, let $\mathcal{C}''$ be the code of length $n+1$ generated by $H' = (H|s^{\mathrm{T}})$. Since $Hx^{\mathrm{T}} = s^{\mathrm{T}}$, we have $H' \cdot (x|1)^{\mathrm{T}} = 0^{\mathrm{T}}$. Therefore, the SD problem can be solved by finding a vector $e \in \mathcal{C}''$ with $\mathrm{wt}(e) \leq t_0 + 1$.
The other direction is obvious since the solution of the general decoding and the SD problem are low-weight vectors in the respective codes. □

For these two problems, the corresponding decision problems were proved to be NP-complete in 1978 [11], but only for binary codes. In 1994, A. Barg proved that this result holds for codes over all finite fields ([6, in Russian] and [7, Theorem 4.1]).

In [34], Finiasz proved the NP-completeness of the following variant of this problem:

Problem 2.4.4 (Goppa Parameterized Syndrome Decoding (GPSD))**.** *Given a binary matrix H of size $2^m \times r$ and a syndrome s, decide whether there exists a word x of weight r/m such that $Hx^T = s^T$.*

This variant is interesting in the context of this thesis since it proves that the SD problem remains NP-complete for those parameters that are used for Goppa codes: a binary Goppa code of length $n = 2^m$ and co-dimension r with an error-correcting capability of $t = r/m$.
Note that this result only regards the parameters and does *not* prove that the problem remains hard when H corresponds to a Goppa code (instead of a random code).

The following problem can be used to link the security of Goppa codes to that of random codes.

Problem 2.4.5 (Goppa code Distinguishing (GD))**.** *Given an $r \times n$ matrix H, decide whether H is the parity check matrix of a Goppa code.*

In 2010, Faugère et al. [32] showed that Goppa codes of very high rate $R = k/n$ can be distinguished from random codes, rendering Dallot's security proof [29] of the CFS signature scheme invalid. However, the distinguisher does not pose a direct threat to existing cryptosystems since it does not constitute a technique to decode or reveal the secret key.

2.4.2. Encryption schemes

Since McEliece's first proposal [55] in 1978, several encryption schemes have been published in code-based cryptography. In this section, we will briefly describe the three encryption schemes relevant for this thesis: McEliece [55], Niederreiter [70], and HyMES [84] by Sendrier and Biswas.
In the following, let G be a $k \times n$ generator matrix for an (n, k, t) Goppa code $\mathcal{C}$, H be a corresponding $r \times n$ parity check matrix, and s a vector of length r. Let the message m be a vector of length k, and φ a bijective function mapping an integer to a word of length n and weight t. All matrices and vectors are defined over a finite field $\mathbb{F}_q$, where q is a prime power.

McEliece

The McEliece public-key encryption scheme was presented by R. McEliece in 1978 [55]. The original scheme uses binary Goppa codes, for which it remains unbroken (with suitable parameters), but the scheme can be used with any class of codes for which an efficient decoding algorithm is known.

Algorithm 1 The McEliece cryptosystem

Notation for Algorithm 1:
- P : An $n \times n$ random permutation matrix
- S : A $k \times k$ invertible matrix
- $\mathcal{D}_G$: A decoding algorithm for the underlying (n, k, t) code $\mathcal{C}$

Encryption $\mathbf{Enc}^{\mathbf{McEliece}}$
INPUT: Message $m \in \mathbb{F}_q^k$ and random seed $r \in \{0,1\}^*$
OUTPUT: Ciphertext $c \in \mathbb{F}_q^n$

$\widehat{G} \leftarrow SGP$
$e \leftarrow \mathrm{PRG}(r)$, such that $\mathrm{wt}(e) = t$
$c \leftarrow m\widehat{G} + e$

Return c

Decryption $\mathbf{Dec}^{\mathbf{McEliece}}$
INPUT: Ciphertext c
OUTPUT: Message m

$\widehat{c} \leftarrow cP^{-1} = mSG + eP^{-1}$
$mSG \leftarrow \mathcal{D}_G(\widehat{c})$
▷ Let $J \subseteq \{1, \ldots, n\}$ be a set such that $G_{\cdot J}$ is invertible
$m \leftarrow mSG \cdot G_{\cdot J}^{-1} \cdot S^{-1}$

Return m

Niederreiter

In 1986, H. Niederreiter proposed a cryptosystem [70] which can be seen as dual to the McEliece scheme. Originally, the scheme uses GRS codes, but this has been shown to be

insecure by Sidelnikov and Shestakov [87]. The scheme uses a parity check matrix of a (usually binary Goppa) code to compute the syndrome of the message, which serves as the ciphertext. Even though the Niederreiter cryptosystem has been proven equally secure as the McEliece system [52], it is threatened by broadcast attacks (see Section 3).
In contrast to the McEliece scheme, where the codeword contains the message and the error vector is used to hide it, the Niederreiter scheme encodes the message into the error vector. Since the underlying Goppa code can only correct a certain number $t < n$ of errors, a function φ is used which maps the message to a word of length n and weight t, which is then encrypted.

Algorithm 2 The Niederreiter cryptosystem

Notation for Algorithm 2:
$\mathcal{D}_H$: A decoding algorithm for the underlying $(n, k = n - r, t)$ code $\mathcal{C}$

Encryption $\mathbf{Enc^N}$
INPUT: Message $m \in \mathbb{F}_q^l$
OUTPUT: Ciphertext $c \in \mathbb{F}_q^r$

$c \leftarrow H \cdot \varphi(m)^{\mathrm{T}}$

Return c

Decryption $\mathbf{Dec^N}$
INPUT: Ciphertext c
OUTPUT: Message m

$m \leftarrow \varphi^{-1}(\mathcal{D}_H(c))$

Return m

HyMES

The HyMES (Hybrid McEliece Encryption Scheme) cryptosystem [84] developed by Biswas and Sendrier is a hybrid between the McEliece and Niederreiter cryptosystems. It increases the efficiency of the McEliece scheme by encoding part of the message into the error vector. While in the usual scenario this scheme is as secure as the original McEliece scheme, we will show in Section 3 that it is vulnerable to a broadcast attack.
The HyMES scheme works as follows: The message m is split into two parts $m = (m_1|m_2)$. The first part m_1 corresponds to the message in the original McEliece scheme, while the second part is encoded into a word of weight t and serves as the error vector $e = \varphi(m_2)$. There are many possible encoding functions φ, e.g. enumerative encoding or encoding into regular words, but the choice of φ is not relevant in our context.

2.4.3. Signature and Identification schemes

In this section, we recall the CFS (Courtois-Finiasz-Sendrier) signature scheme [28] which is the basis for our QD-CFS signature in Section 5.2. In addition to that, we will briefly describe the Stern identification (ID) scheme. This scheme and its variant, the CVE

Algorithm 3 The HyMES cryptosystem

Notation for Algorithm 3:
P : An $n \times n$ random permutation matrix
S : A $k \times k$ invertible matrix
$\mathcal{D}_G$: A decoding algorithm for the underlying (n, k, t) code $\mathcal{C}$

Encryption $\mathbf{Enc}^{\mathbf{HyMES}}$
INPUT: Message $m \in \mathbb{F}_q^{k+l}$
OUTPUT: Ciphertext $c \in \mathbb{F}_q^n$

$\widehat{G} \leftarrow SGP$
$m_1 \leftarrow \mathrm{MSB}_k(m)$
$m_2 \leftarrow \mathrm{LSB}_l(m)$
$c \leftarrow m_1\widehat{G} + \varphi(m_2)$

Return c

Decryption $\mathbf{Dec}^{\mathbf{HyMES}}$
INPUT: Ciphertext c
OUTPUT: Message m

$\widehat{c} \leftarrow cP^{-1} = m_1SG + \varphi(m_2)P^{-1}$
$mSG \leftarrow \mathcal{D}_G(\widehat{c})$
▷ Let $J \subseteq \{1, \ldots, n\}$ be a set such that $G_{\cdot J}$ is invertible
$m_1 \leftarrow mSG \cdot G_{\cdot J}^{-1} \cdot S^{-1}$
$m_2 \leftarrow \varphi^{-1}(c - m_1\widehat{G})$

Return $(m_1|m_2)$

(Cayrel-Véron-El Yousfi Alaoui), serve as illustrations in Section 4.2 in the context of partial knowledge.

CFS signatures

The CFS signature scheme is one of the first code-based signature schemes. Advantages of the scheme are the small signature size (for parameters $(m, t) = (15, 12)$ approx. 180 bit) and its very fast verification procedure. The main drawbacks are the large public key size (which will be addressed in Section 5.2) and a slow signature generation.

A generic technique to build signatures from (public-key) encryption schemes is to essentially reverse the process. A signature is created as follows:

- Compute the hash value $h = h(m)$ of the document m which is to be signed.
- Treat h as a ciphertext and decode it into e.
- The signature for m is e.

This signature can easily be verified by using the public key to encode e and checking whether the result is $h(m)$.

Unfortunately, this technique can not be directly applied in code-based cryptography since for most codes the vast majority of hash values are not decodable. In CFS, this problem is addressed by appending an integer i as a counter to the hashed message and hashing again, $h = h(h(m)|i)$. In the original paper, this counter is a sequentially increasing number, while Dallot [29] proposed a random counter instead. The algorithm repeats this computation until the resulting hash value is decodable. The signature consists of (e, i), where e is the result of decoding $h(h(m)|i)$. More formally, the CFS signature scheme consists of the following algorithms:

- Keygen: Choose an appropriate (n, k, t) code over $\mathbb{F}_q$ defined by a public parity-check matrix H with a private decoding algorithm $\mathcal{D}_H$. Let $h()$ be a public hash function.
- Sign: Let $m \in \{0,1\}^*$ be the message to sign. Find $i \in \mathbb{N}$ (either sequentially or by random sampling) such that $s = h(h(m)|i)$ is a decodable syndrome. Using the decoding algorithm $\mathcal{D}_H$, find $e \in \mathbb{F}_q^n$ of weight $\mathrm{wt}(e) \leq t$ such that $He^{\mathrm{T}} = s^{\mathrm{T}}$. The signature is the pair (e, i).
- Verify: Let (e, i) be a purported signature for message m. Compute $s = h(h(m)|i)$, and accept iff $\mathrm{wt}(e) \leq t$ and $He^{\mathrm{T}} = s^{\mathrm{T}}$.

The expected number of iterations which are required to find a decodable codeword is the inverse of the code density (i.e. the fraction of decodable words). For binary Goppa codes, the code density is approximately $1/t!$. Therefore, it is necessary to choose parameters with a small value of t. Typical CFS parameters are $n = 2^{15}$ and $t = 12$.

Stern identification scheme

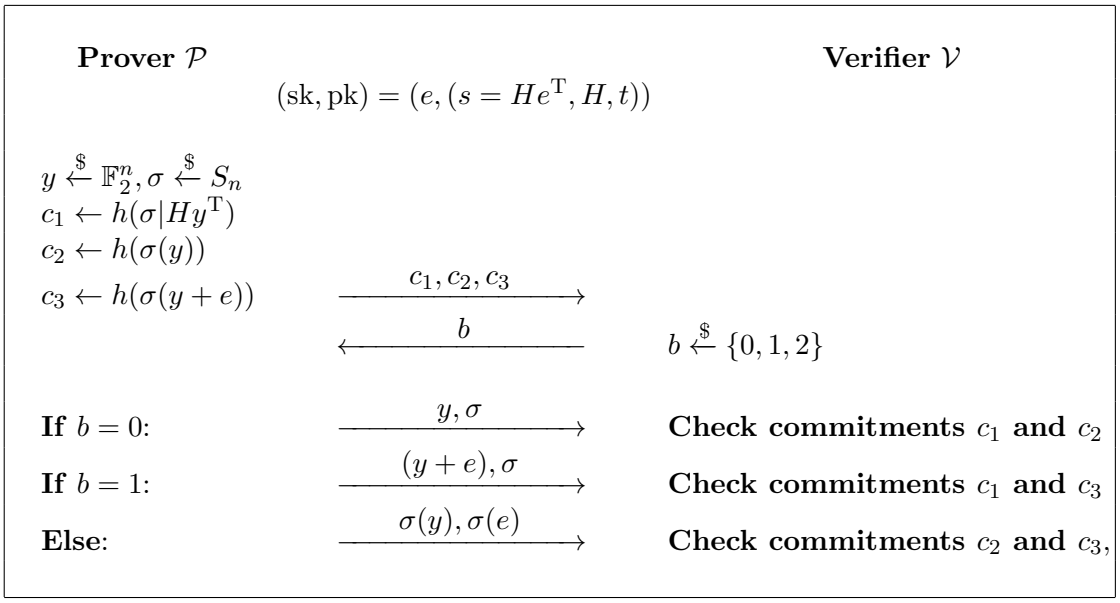
Stern presented a code-based identification scheme [89] in 1993. This zero-knowledge scheme is the basis for many subsequent improvements and variants, amongst them the CVE which we use in Section 4.2. We will briefly describe the scheme in order to illustrate the working principle.

The idea of the scheme is that the owner of a secret key e proves his identity by showing that he knows the secret, without giving away any information about it (hence zero-knowledge). The secret key holder protects the secret e by using two techniques: the transformation by means of a permutation, and the addition of a random vector. Since a cheater has a 1/3 chance of successfully completing the protocol without the knowledge of the secret, the protocol has to be run several times to detect cheating provers. The security of the construction relies on the hardness of the general decoding problem, that is, on the difficulty of determining the preimage (with respect to left hand multiplication by H) e of $s^T = He^T$. In Figure 2.1, $h()$ denotes a hash function, H the parity check matrix on a random binary (n, k, t) code and S_n the symmetric group of degree n.

2.4.4. Hash functions

Hash functions play an integral role in cryptography. While code-based cryptosystems can usually be combined with any hash function, code-based hash functions are especially

Figure 2.1.: Identification protocol



interesting in our context since they can often be attacked with similar techniques as code-based cryptosystems. In this section, we will present the FSB hash function which is referred to in Section 4 of this thesis.

Fast Syndrome-Based (FSB) hash function

The FSB hash function [5] was proposed in 2003 and became a second-round SHA-3 candidate. It is based on the NP-complete *regular syndrome decoding* problem which is a modification of the usual SD problem for regular words. FSB is based on a compression function and uses the Merkle-Damgård domain extender [58, 31] to construct the hash function. Hence, attacks are usually done on the underlying compression function. Table 2.1 shows different parameters sets proposed in [5].

Table 2.1.: Parameters for the five versions of FSB.

	n	w	r	p	s
FSB_{160}	$5 \cdot 2^{18}$	80	640	653	1120
FSB_{224}	$7 \cdot 2^{18}$	112	896	907	1568
FSB_{256}	2^{21}	128	1024	1061	1792
FSB_{384}	$23 \cdot 2^{16}$	184	1472	1483	2392
FSB_{512}	$31 \cdot 2^{16}$	248	1984	1987	3224

The working principle of FSB is as follows:

- In every iteration, the compression function outputs an r bit vector.
- The input are $s-r$ bits from the document, concatenated with either the initialization value (in the first iteration) or the output from the previous iteration.
- If necessary, a padding is applied to the input of the last round, and the Whirlpool compression function is applied to compute the final hash value of the document.

Regular words

Before the s input bits are handed to the compression function, they are transformed into a regular word. This is done in order to increase the efficiency of the hash function (see Section 2.4.5). Regular words are defined as follows.

Definition 2.4.6 (Regular words)**.** A regular word of length n and weight t over a finite field $\mathbb{F}_q$ is a vector $e \in \mathbb{F}_q^n$ consisting of blocks of size n/t, where each block contains exactly one non-zero entry.

Remark 2.4.7. *Bernstein et al. showed how to use the regular words structure to improve the efficiency of ISD [17].*

2.4.5. Constant weight encoding functions

Constant weight encoding (CWE) functions are maps from arbitrary vectors to vectors of constant weight. The length of input and output can be variable or fixed. CWE functions are used in the Niederreiter and HyMES cryptosystems in order to encode the message (or part of the message) into a vector of length n and weight t. This allows the message to be decoded uniquely. In the FSB hash function (see Section 2.4.4), a CWE is applied to the input of the compression function in order to ensure that collision and other attacks are difficult. Also, CWE functions (or rather, their inverse) can be used for compression if the data contains vectors of constant weight.
In this section, we will present two CWE functions: Enumerative encoding and encoding into regular words.

Enumerative encoding

For fixed input and output length, enumerative encoding is the most efficient encoding function in terms of the length of the vectors. For given integers n and t and a finite field $\mathbb{F}_q$, it maps

$$\varphi : \mathbb{F}_q^{l_q} \to \mathcal{W}_{n,t,q},$$

where $l_q = \lfloor \log_q(\binom{n}{t}(q-1)^t) \rfloor$ and $\mathcal{W}_{n,t}$ is the set of all vectors of length n and weight t over $\mathbb{F}_q$. Binary enumerative encoding has been described in different sources, e.g. [13, Section 5.1]. In order to use this encoding technique with cryptosystems which are defined of non-binary fields $\mathbb{F}_q$, the encoding algorithm should be generalized to $\mathbb{F}_q$ as well. We propose the following modification (even though we are not aware of other publications

proposing this method, we do not claim that we are the first to suggest it). First note that we can use the error *positions* to encode $\binom{n}{t}$ different numbers, and for each of these combinations of positions, $(q-1)^t$ different *values*. Let $\mathbf{Enc}_2()$ denote binary enumerative encoding $\mathbb{F}_2^{l_2} \to \mathcal{W}_{n,t,2}$, our algorithm thus works as follows:

Algorithm 4 Enumerative encoding over $\mathbb{F}_q$

INPUT: Integers n, t, q and a vector $v \in \mathbb{F}_q^{l_q}$
OUTPUT: Vector $e \in \mathcal{W}_{n,t,q}$

$i \leftarrow \sum_{j=1}^{l_q} v_j q^{j-1}$
$d \leftarrow \lfloor i(q-1)^{-t} \rfloor$
$e \leftarrow \mathbf{Enc}_2(d)$
$r \leftarrow i - d(q-1)^t$

$p \leftarrow t-1$
for j from 1 to n **do**
  if $e_j \neq 0$ **then**
    $e_j \leftarrow \lfloor r(q-1)^{-p} \rfloor + 1$
    $r \leftarrow r - (e_j - 1)(q-1)^p$
    $p \leftarrow p - 1$
  end if
end for

return e

Regular words encoding

While enumerative encoding is most efficient in terms of the output length, it is not very fast since the evaluation of $\mathbf{Enc}_2()$ requires the computation of t binomial coefficients. Even though there are techniques to compute these faster than the naïve approach, the complexity remains high.
A different encoding method is encoding into regular words. As noted above, regular words of length n and weight t consist of blocks of size n/t and weight 1. Depending on the context, there are different encoding techniques. A simple binary algorithm works as follows:

- For each block of size n/t, get the next $\lfloor \log_2(n/t) \rfloor$ bits from the input.
- Convert the input bits to an integer $i \in \mathcal{I}_{n/t}$.
- Set the i-th bit of the current block to 1 and all others to 0.

This algorithm can be generalized to $\mathbb{F}_q$ by using $\lfloor \log_q(qn/t) \rfloor$ input bits and setting the bit to the respective value in $\mathbb{F}_q$.

2.5. Cryptanalysis

This section covers those cryptanalytic techniques which are relevant for this work. Attacks against cryptosystems are often classified into either *critical* or *non-critical* attacks.

The former exploit structural weaknesses in the construction and cannot be countered efficiently by simply increasing the key length. Therefore, special attention is required in the construction process to prevent these attacks. The latter attacks are exponential in the security parameter and can therefore be prevented by choosing suitable parameters. We conclude this topic with a section on attacker models.

2.5.1. Critical attacks

In the following, we briefly present critical attacks against code-based cryptosystems which are relevant in the context of this thesis. We conclude with an overview on the vulnerability of the McEliece, Niederreiter, and HyMES cryptosystems (when used without semantic conversions) against each of these attacks. More details can be found in [64].

Known partial plaintext

This type of attack applies when an attacker knows part of the plaintext he attempts to reveal. This scenario arises in many applications, e.g. standardized emails or electronic forms.
The complexity of decoding the ciphertext decreases exponentially with every known bit. For example, attacking a ciphertext encrypted with McEliece using parameters (n, k) and knowing k_l bits is equivalent to attacking a McEliece ciphertext encrypted using parameters $(n, k - k_l)$. See [63, 23, 48] for more details.

Message-resend and related-message

A message-resend condition is given if the same message is encrypted and sent twice (or several times) to the same recipient. If the subsequent messages are related to the first by a known relation, it is called a related-message condition.
In the case of McEliece and HyMES, these conditions can be detected by observing the Hamming weight of the sum of two ciphertexts. By comparing the two ciphertexts, an attacker can identify those bits where

- with high probability, neither ciphertexts contains an error, or
- with certainty, exactly one contains an error.

This allows to recover the ciphertexts in negligible time [63].

Sidelnikov-Shestakov attack

In 1992, Sidelnikov and Shestakov proposed an attack [87] on Niederreiter's cryptosystem using GRS codes which aims to recover an alternative private key from the public key. They take advantage of the fact that the (secret) parity check matrix H of a GRS code consists of entries $H_{ij} = z_i a_i^j$. Sidelnikov and Shestakov concluded that each entry of the public-key parity check matrix H' can be expressed by a polynomial in a_i. From this observation they were able to derive a system of polynomial equations whose solution yields the private key. Using this method, it is possible to decipher the message in polynomial time.

Figure 2.2.: Overview on the vulnerability of the McEliece, Niederreiter, and HyMES cryptosystems (without semantic conversions) using Goppa codes against different critical attacks.

		McEliece	Niederreiter	HyMES
Plaintext	Broadcast (Section 3.1 and [65])	no	*	*
	Known partial [63]	*	*	*
	Message-resend [63]	*	no	*
	Related-message [63]	*	no	*
Ciphertext	Chosen	*	*	*
	Lunchtime	*	*	*
	Adapt. chosen [92]	*	*	*
	Reaction [48]	*	*	*
	Malleability [48]	*	no	*

Reaction attack

This attack can be considered a stronger version of a chosen ciphertext attack. Instead of receiving the decrypted ciphertexts from the oracle, the attacker only observes the reaction of the oracle. Usually, this means whether the oracle was able to decrypt the ciphertext. In the context of side-channel-attacks, this can also mean observing decryption time, power consumption etc.
In one of the easiest variants, the attacker flips individual bits of the ciphertext he attempts to decode, and observes whether the oracle is able to decrypt it. If that is the case, the bit corresponds to an error bit (McEliece / HyMES). In the Niederreiter case, the same can be achieved by adding columns of the parity check matrix to the syndrome.

Malleability

A cryptosystem is vulnerable to the malleability of it's ciphertexts if it is possible for an attacker to create new valid ciphertexts from a given one and if the new ciphertexts decrypts to a cleartext related to the original message. This property is relevant in many scenarios, e.g. bank transactions, where an attacker could change the amount of money transferred.

2.5.2. Non-critical attacks

This section covers three important non-critical attacks: Information-Set Decoding (ISD), the Generalized Birthday Algorithm (GBA), and the Support Splitting Algorithm (SSA). ISD and GBA are decoding attacks, i.e. they attempt to solve the general decoding or the SD problem. Due to the way the algorithms work, ISD is usually more efficient if the decoding problem has only one or a small number of solutions, while GBA is often better to find one out of many possible solutions. The SSA attempts to recover the secret key by searching for codes with known structure which are permutation equivalent to the given

code.

A principle that is used in most ISD-like attacks and on which GBA attacks are based is the birthday paradox.

The Birthday Paradox

The name is due to the fact that a surprisingly small number of people are required such that the probability of at least two of them having the same birthday is greater than 50% (this number is 23). In the context of ISD, this principle is used in the following way: In order to find p columns of a matrix H that sum up to a given vector s, we build two lists. L_1 contains sums of $p/2$ columns of H, and $L_2 = \{s - x : x \in L_1\}$. Finding a common entry between these lists corresponds to the birthday situation, hence we can expect to quickly find a large number of these (so called) collisions. GBA attacks use a generalization of this principle by allowing more than two lists.

Information Set Decoding

ISD algorithms are among the most efficient generic attacks against code-based cryptosystems like the McEliece encryption scheme, the CFS signature scheme [28], the FSB hash function [5], and others. ISD algorithms solve the general decoding problem, i.e. decoding codewords with errors. More specifically, if m is a plaintext and $c = mG+e$ is a ciphertext, where e is a vector of weight t and G a generator matrix, then ISD algorithms take c as input and recover m (or, equivalently, e). Since $s^T := Hc^T = H(mG + e)^T = He^T$, the problem is often formulated using a parity check matrix as in Problem 2.4.2.

A basic version of an ISD algorithm works as follows: a random permutation P is applied to H in the hope that all columns corresponding to error positions in e are moved to the left hand side of the matrix (in the first $n - k$ columns). Then Gaussian elimination is used to transform H into the form $H' = [I_{n-k}|R]$, where I_{n-k} is the $(n - k) \times (n - k)$ identity matrix, and the row operations are performed on s as well to get s'. If s' has a weight not exceeding t, the algorithm succeeds; we can read off the error positions from s' and get $e = P^{-1}[s'|0^k]$, where 0^k is the zero vector of size k. Otherwise, the algorithm restarts.

Most advanced ISD versions make use of the birthday paradox: they allow a certain (usually small) number p of errors in the last $k - l$ columns of H. Then lists of column sums of H are used to find these error positions. If we split the right hand part of H into $[H_1|H_2]^T$, and write $e = [e_1|e_2]$, then we search for a vector e_2 of weight p such that $s^T - H_2e_2^T$ has weight $t - p$, and the non-zero positions of $s^T - H_2e_2^T$ show the remaining $t - p$ error positions.

Over the years, there have been many improvements and generalizations of ISD-like attacks. The most important of these are listed in Table 2.2, together with their respective binary work factor to decode a $(1024, 524, 50)$ Goppa code (these are the original McEliece

parameters).

Table 2.2.: Complexity of ISD algorithms against $(1024, 524, 50)$ McEliece cryptosystem. This table is from [69].

Year	Algorithm	Log of binary work factor
1986	Adams-Mejier [3]	80.7
1988	Lee-Brickell [49]	70.89
1989	Stern [88]	66.21
1994	Canteaut-Chabanne [21]	65.5
1998	Canteaut-Chabaud [22]	64.1
2008	Bernstein-Lange-Peters [15]	60.4
2009	Finiasz-Sendrier [36]	59.9

Ball-collision decoding [16], published in 2011, has a complexity of 61.6 bit against these code parameters (using $(p_1, p_2, q_1, q_2, l_1, l_2) = (4, 4, 1, 1, 21, 22)$). While this attack is more efficient than Finiasz and Sendrier's lower bound for large codes — the authors use the examples $n = 6624$ and $n = 30,332$ — it shows to be less efficient for smaller codes.

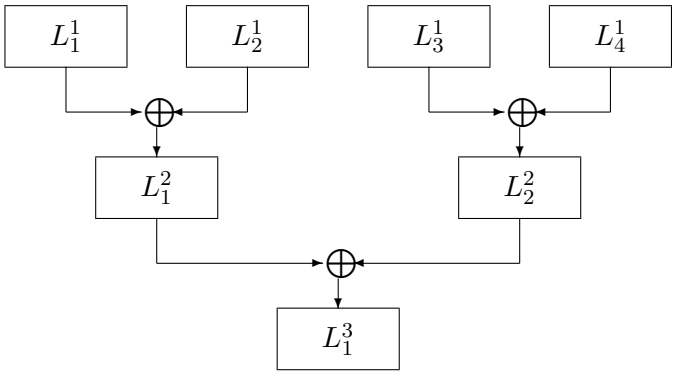
Generalized Birthday algorithm

Generalized Birthday algorithms are used for some of the most efficient attacks against code-based cryptosystems. They have been proposed by Wagner in 2002 [97], and they are named after the famous birthday paradox which allows to quickly find common entries in lists. In 2009, Minder and Sinclair proposed a further generalization of this algorithm [60] which allows to trade-off time and space efficiency. In this section we will review the basic principles of Generalized Birthday Attacks (GBA).

GBA attempts to solve the SD problem stated above (Problem 2.4.2 on page 25), i.e. for given parity check matrix H, syndrome s and integer t, to find a vector e of weight t such that $He^{\mathrm{T}} = s^{\mathrm{T}}$. In other words, find a set of columns of H whose weighted sum equals the given syndrome. For easier demonstration of the algorithm, let $t = 2^a$ for some integer a. If this is not the case, the algorithm still works with a slight loss of efficiency. Let H be the parity check matrix of a $[n, k]$ code, i.e. H has size $r \times n$.
First, the algorithms sets up t lists $L_1^1, \ldots, L_t^1$, each of which consists of every column of H (in the q-ary case, also all non-zero multiples of these columns). Each list therefore consists of $(q-1)n$ vectors in $\mathbb{F}_q^r$.
In the next step, pairs of lists are merged into new ones: For $i \in [1, t/2]$, the lists L_{2i-1}^1 and L_{2i}^1 are merged to form a new list L_i^2, with $L_i^2 = L_{2i-1}^1 \oplus L_{2i}^1$. This means that the new list is created by considering all pairs of one element from L_{2i-1}^1 and one from L_{2i}^1, and adding their sum to L_i^2. If an entry occurs more than once in a new list, it is removed to minimize the size of the lists. The maximum size of L_i^2 is hence $((q-1)n)^2$.
Following this, the lists L_i^2 are merged to form L_i^3, and this merging step is repeated until exactly one list remains. If this list contains the vector s, then the algorithm was

successful, and from the columns of H which were added to form s we can read of the solution e. Often, a slight modification is introduced: The vector s is subtracted from each entry in the last list L_t^1, and the algorithm attempts to find a combination of entries that sum up to zero.

Figure 2.3.: Illustration of the GBA for $a = 2$.



In fact, the final list contains *all* possible solutions. The number of solutions can be quite large for certain applications, e.g. for collision attacks on signature schemes. This fact can be used to further improve the attack, as shown by Wagner (and in a more general way, by Minder and Sinclair):
After each merge step, the lists are being reduced in size. This is done by forcing a certain, increasing number of bits to zero — i.e. to delete all list elements which are non-zero on at least one of the positions forced to zero. Each forced bit reduces the expected list size by a factor of q, and the number of bits that are forced to zero in each step can be optimized with respect to the parameters $[n, r]$ and t. The reduction of the list size is an improvement in space — less data needs to be stored — as well as in time, since there are less elements that need to be manipulated.
The time complexity of the GBA is in $\mathcal{O}(2^{a+u})$, where 2^u is the maximum expected list length and $t = 2^a$. See [60] for a detailed proof.
This fact is the reason why GBA attacks are usually most efficient when there are a large number of solutions. For example, an unpublished attack by D. Bleichenbacher against the CFS signature scheme [28] was very efficient and required an increase in the CFS parameters to remain secure (the attack is described in [36]).

Remark 2.5.1. *There are (at least) two possible optimization techniques:*

1. *Depending on the setup of the initial lists L_i^1, some of the lists L_i^j can be identical and do not need to be computed or stored twice (e.g. if $L_1^1 = L_3^1$ and $L_2^1 = L_4^1$, then $L_1^2 = L_2^2$)*

2. *Instead of computing all lists on one level of the tree before continuing to the next level, we can compute the lists in a "depth-first" way:*

- *Set up* L_1^1 *and* L_2^1
- *Compute* L_1^2 *and delete* L_1^1 *and* L_2^1
- *Set up* L_3^1 *and* L_4^1
- *Compute* L_2^2 *and delete* L_3^1 *and* L_4^1
- *Compute* L_1^3 *and delete* L_1^2 *and* L_2^2
- *...*

This allows to reduce the required memory since only approximately one list per level of the tree needs to be stored. See [14], for example, where this technique was used in an attack on the FSB hash function.

Decoding one out-of-many

In cryptography, different scenarios exist which are used to assess the security of a given cryptographic scheme (see Section 2.5.3). Typically, the attacker is given exactly one instance of the problem and attempts to break it, e.g. decode a given ciphertext. However, there are some situations where several or even an unlimited number of instances are available and the attacker is satisfied with breaking only one of them. For example, in the CFS signature scheme the attacker can generate a large number of hashed documents using different counters and needs to decode only one.

For GBA, Bleichenbacher developed an (unpublished) attack that takes this scenario into account and saves a factor of approximately $\sqrt{N}$, where N is the number of instances. However, the improvement factor is upper bounded since there is an overhead cost involved (e.g., every instance needs to be read at least once), so the maximum improvement factor is $T^{1/3}$, where T is the original cost of the attack; in other words, the running cost exponent can be decreased to $2/3$ at most.

In a recent paper, Sendrier [83] showed how to make use of the "Decoding One Out-of-Many" scenario (dubbed DOOM) to improve the efficiency of an ISD attack. The idea is similar to Bleichenbacher's in that it improves the collision search step of ISD, in which a common element (collision) of two lists is searched. In this step, the set of available syndromes $\mathcal{S}$ is used to increase the size of one of the lists by a factor of approximately $|\mathcal{S}|$, thereby increasing the success probability. This improvement decreases the running cost exponent to $2/3 + c$ (with small positive c), which is close to the theoretical optimum of $2/3$.

Support splitting algorithm

The SSA was presented by Sendrier [81] in 2000. This algorithm decides the question whether two given codes are permutation equivalent, i.e. one can be obtained from the other by permuting the coordinates. While this problem is not NP-complete, is has been proved to be at least as hard as the graph isomorphism problem [76].

To solve this problem, the SSA makes use of invariants and signatures (not to be confused with the digital signatures or the signature of a dyadic matrix). An invariant is a property of a code that is invariant under permutation, while a signature is a local property of a code and one of its coordinates. The difficulty of using invariants and signatures to decide whether two codes are permutation equivalent is that most invariants are either too coarse (i.e. they take the same value for too many codes which are not permutation equivalent) or the complexity to compute them is very high.
The SSA solves this issue by starting with a coarse signature and adaptively refining it in every iteration. The total complexity of the SSA applied to a code is polynomial in the length of the code and exponential in the dimension of its hull $\mathcal{H}(\mathcal{C}) = \mathcal{C} \cap \mathcal{C}^{\perp}$, i.e. the intersection of the code with its dual.

The SSA is not a direct attack against a code or a cryptosystem, but it can be used to recover the secret. See Section 5.1.4 (page 108) for this.

2.5.3. Attack models

There are several attack models proposed in the literature. These models describe what capabilities an assumed attacker has. Some of the more common models are:

- Ciphertext-only
- Known-plaintext
- Chosen-plaintext
- Chosen-ciphertext

There exist many variants of these models. For example, adaptive chosen-ciphertext attacks and lunchtime attacks are variants of chosen-ciphertext attacks.

In general, this thesis focusses on ciphertext-only attacks. This means that an attacker has access only to one or several ciphertexts (in addition to any public information, which we consider to be available). An encryption scheme is considered to be broken if the attacker finds the corresponding plaintexts or the private keys. The model is defined correspondingly for signature schemes, where the attacker needs to find signatures for one or more given messages.

There are two exceptions to this: Firstly, Section 3 considers a broadcast scenario. This is a variant of the ciphertext-only model in which the given ciphertexts correspond to the same or to very similar messages. Secondly, in Section 4.2 we assume that the attacker has some well-defined additional information that can be exploited in an attack.

Security level

Let $\mathcal{C}$ be an $[n, k, d]$ code over $\mathbb{F}_q$. When this code is used in a cryptographic scheme, the properties of the scheme – e.g. key size, message length, security – are influenced by these

four parameters. In order to measure the security of a cryptographic scheme, we introduce the *security level*:

Definition 2.5.2 (Security level)**.** Choose a code-based encryption scheme (e.g. McEliece), let $\mathcal{C}$ be an $[n, k, d]$ code over $\mathbb{F}_q$, and C be the set of valid ciphertexts corresponding to this scheme with the above parameters. We define the security level s of $\mathcal{C}$ against a certain attack as $s = \log_2(o)$, where o is the expected number of arithmetic operations of a successful application of this attack against a ciphertext chosen uniformly at random from C. When the attack is not specified, the definition refers to the best known attack (i.e. the minimum over all known attacks).
We can extend this definition to the other types of cryptographic applications used in Section 4.2.3 (the CVE identification scheme, page 77) and Section 5.2.3 (the QD-CFS signature scheme, page 113). In the former example, C is the set of valid secrets e from which Alice has chosen her private key; in the latter example, C refers to the set of valid hash values of documents in which a collision or first/second preimage is to be found.

3. Critical Attacks

While there is strong evidence that cryptosystems like McEliece and Niederreiter are secure if appropriate parameters are selected, they have certain weaknesses when used without semantic conversions. Examples include message-resend, related-message, reaction, and known-ciphertext attacks (see Section 2). These have been analyzed in the context of code-based cryptography (see [48] or [63] for an overview on these attacks).
A scenario that has not been considered yet in code-based cryptography is a broadcast scenario, where the same message is (or similar messages are) sent to different users, encrypted with the respective public keys. It is especially relevant for practical implementations since it appears often in real-world applications, for example:

- Standardized emails
- Electronic banking
- Purchase orders and other business transactions

It is therefore necessary to understand the implications of broadcast attacks and how to prevent them.

Our contributions

We have developed a broadcast attack that attempts to recover the secret message. In the following, we show how and under what conditions this attack can be mounted against two code-based encryption schemes: the Niederreiter and the HyMES cryptosystem. We show that if the public keys corresponding to the intercepted messages are independent from each other, we expect to require no more than N_r recipients to run the attack:

$$N_r := \left\lceil \frac{n+2}{r} \right\rceil,$$

where n and $r = n - k$ denote the code parameters of the users' secret keys. That means that with near certainty, $N \geq N_r$ recipients suffice to run the attack. Table 3.1 shows the required number of recipients for selected parameter sets taken from Bernstein et al. [13] and Biswas [18]. In most cases, a very small number of messages — usually in the order of 10 — is sufficient to completely break the schemes. We achieve this by combining the intercepted information into a large set of linear equations until the system has full rank and can be solved.
In addition to that, we treat the cases when the attacker receives fewer messages than required for this attack, and when the cleartexts are related, instead of identical. In the former case, the attack complexity is higher compared with the broadcast attack before,

Table 3.1.: Number of required recipients for Niederreiter and HyMES parameter sets. The first column shows the cryptosystem for which the parameters were developed.

Cryptosystem	n	k	q	Number of recipients N_r
Niederreiter([13])	2048	1696	2	6
	2048	1608	2	5
	4096	3832	2	16
	4096	3556	2	8
HyMES([18])	1024	524	2	3
	2048	1608	2	5
	2048	1696	2	6
	4096	3604	2	9
	8192	7815	2	22

but lower than a generic attack: after setting up the linear equation system, the attacker runs an ISD attack, the complexity of which is smaller the more messages are intercepted. In the latter case, more messages are required to perform the broadcast attack:

$$N_r' := \left\lceil \frac{n+2}{r-u} \right\rceil,$$

where u denotes the number of bits where not all messages are identical to each other.
While many code-based cryptosystems use certain classes of codes, e.g. binary Goppa codes, our attack is completely independent of this choice and solves the underlying problem directly. That means, no matter what class of codes is used, the attack complexity cannot be greater than our results; it might be possible, though, to achieve even better results against certain code classes by exploiting the structure of the code. To illustrate our results, we apply our broadcast attack implementation against the Niederreiter cryptosystem in the FlexiProvider package[1], recovering the message in only a few seconds to minutes (see Table 3.2 on page 54). We conclude this section with a discussion on possible countermeasures and provide a CCA2-secure version of the Niederreiter cryptosystem using the Kobara-Imai conversion.

Related work

In 1988, J. Håstad [43] presented an attack against public key cryptosystems. This attack was originally aimed at the RSA cryptosystem, when a single message is sent to different recipients using their respective public keys. Håstad showed how to recover the message in this broadcast scenario. While this result has been known for a long time, this type of attack has not been considered for cryptosystems based on error-correcting codes.
In [77], Plantard and Susilo describe a broadcast attack against lattice-based cryptosystems. While the scenario is similar to Håstad's and ours, the attack works differently since the lattice-based cryptosystem is not based on the same underlying problem as ours.

[1] `http://www.flexiprovider.de/`

A recent paper [73] by Pan and Deng presents a broadcast attack against the NTRU cryptosystem. The authors use an algorithm by Ding to solve the Learning with Errors (LWE) problem for their attack.

Outlook and further work

As further work we propose to analyze whether additional structure in the cryptosystem makes it more vulnerable to this attack, resulting in a smaller number of required messages. Examples include the use of QC or QD codes, or the conversion of the message to regular words.

3.1. Broadcast attacks against Niederreiter/HyMES

In this section, we will show how to mount a broadcast attack against the Niederreiter and HyMES schemes. The problem solved by a broadcast attack is the following:

Problem 3.1.1 (Broadcast decoding problem)**.** *Given N ciphertexts c_i of the same message m, encrypted using N corresponding public keys G_i (HyMES) or H_i (Niederreiter), find m.*

Note that the broadcast scenario is very similar to the Low-Exponent-Attack on RSA. In that case, a message m is sent to N users after being encrypted using a (small) exponent e and the recipients' individual moduli $n_1, \ldots, n_N$, where $N \geq e$.

Both the McEliece and the Niederreiter cryptosystem rely on the hardness of the SD problem; this is equivalent to finding a codeword in a certain distance to a given word. The main difference is that in McEliece, the cleartext determines the codeword and the error vector is random, while Niederreiter works essentially vice versa. This difference results in a weakness of the McEliece cryptosystem, the malleability of its ciphertexts: Adding rows of the public key to a ciphertext results in a new valid ciphertext. This is not possible for the Niederreiter cryptosystem.
Another implication is McEliece's weakness to message-resend and related-message attacks. Two ciphertexts of messages m_1 and m_2, where the messages have a known relation (or are identical), allow an attacker to easily find at least k error-free positions, which allows efficient guessing of error bits. See [13, Section 5.3] for more details. The Niederreiter cryptosystem, however, is not vulnerable to these attacks. It is interesting to note that, facing a broadcast attack, the situation is reversed.

3.1.1. Attacking Niederreiter

Recall that φ denotes a function mapping onto vectors of constant weight. In the Niederreiter cryptosystem, we use $\varphi : \mathbb{F}_q^{l_q} \mapsto \mathcal{W}_{n,t,q}$, where $l_q = \lfloor \log_q \binom{n}{t} (q-1)^{\mathrm{T}} \rfloor$ and $\mathcal{W}_{n,t,q}$ is the set of all vectors of length n and weight t over $\mathbb{F}_q$. A Niederreiter ciphertext c is generated by computing

$$c^{\mathrm{T}} = H\varphi(m)^{\mathrm{T}}.$$

Attempting to solve this equation for m can be done by solving the corresponding linear equation system with n variables and r equations:

$$\begin{pmatrix} c_1 \\ \vdots \\ c_r \end{pmatrix} = \begin{pmatrix} h_{11} & \cdots & h_{1n} \\ \vdots & \ddots & \vdots \\ h_{r1} & \cdots & h_{rn} \end{pmatrix} \begin{pmatrix} m'_1 \\ \vdots \\ m'_n \end{pmatrix},$$

where $c = (c_1, \ldots, c_r)$, $\varphi(m) = (m'_1, \ldots, m'_n)$, and h_{ij} are the entries of H.
In a broadcast scenario, we have N ciphertexts c_i, $1 \leq i \leq N$, generated by computing $c_i^{\mathrm{T}} = H_i \varphi(m)^{\mathrm{T}}$ using N public keys H_i, for $1 \leq i \leq N$. Since $\varphi(m)$ is identical in all computations, we can append the matrices $H = \langle H_1, \ldots, H_N \rangle$ and the syndromes $c = (c_1 | \ldots | c_N)$. The number of variables remains constant, whereas the number of equations increases. Some of the new equations might be combinations of existing ones, so the new number of equations can be smaller than Nr. In Section 3.1.3, we will show that the number of redundant equations is actually very small. Since n/r is usually small, we need very few ciphertexts c_i to increase the rank of H to n, at which point we can solve the system. We will compute the expected number of messages required to break the system in Section 3.1.3.
Algorithm 5 summarizes the steps to run this attack.

Algorithm 5 Broadcast attack against the Niederreiter cryptosystem

INPUT: Code parameters (n, k, t), N parity check matrices $H_i \in \mathbb{F}_q^{r \times n}$ and corresponding ciphertexts $c_i \in \mathbb{F}_q^r$ for $1 \leq i \leq N$, a finite field $\mathbb{F}_q$, and a function φ
OUTPUT: Message $m \in \mathbb{F}_q^l$

$H \leftarrow \langle H_1, \ldots, H_N \rangle$
$c \leftarrow (c_1 | \ldots | c_N)$

Solve the linear equation system $Hm'^{\mathrm{T}} = c^{\mathrm{T}}$ over $\mathbb{F}_q$ for $m' \in \mathcal{W}_{n,t,q}$

Return $m = \varphi^{-1}(m')$

There is a different way to describe our attack, seen from another perspective: by adding c_i as the $(n+1)$-th column of H_i for all i, we add $(\varphi(m)|(q-1))$ as a codeword to every code, since

$$(H_i|c_i) \cdot (\varphi(m)|(q-1))^{\mathrm{T}} = H_i \cdot \varphi(m)^{\mathrm{T}} + (q-1)c_i = qc_i = 0.$$

This codeword — and thus the message — can be found by intersecting these new codes. There are several implementations to compute the intersection of codes, e.g. in Magma. However, the approach above is better suited for our purpose since it allows easier understanding of the required number of recipients and of the application of ISD algorithms, as we will show in Sections 3.1.3 and 3.1.4, respectively.

3.1.2. Attacking HyMES

While the HyMES implementation [84] does provide methods for padding, a technique used to prevent attacks that exploit relations between cleartexts and/or ciphertexts, this

method seems to be a placeholder since it does not add any security. Similar to our description of the scheme above, in the broadcast scenario we compute the ciphertexts c_i as

$$c_i = m_1 G_i + \varphi(m_2),$$

where $m = (m_1|m_2)$ and $1 \leq i \leq N$.

Attacking this scheme can be done by finding $\varphi(m_2)$ since this allows to find m_1 by simple linear algebra: As the generator matrices G_i are by definition of full rank, the functions $m_1 \mapsto m_1 G_i$ are injective. Therefore, m_1 is the unique solution of the system $xG_i = c_i - \varphi(m_2)$. It can be found by Gaussian elimination, for example. The problem of finding $\varphi(m_2)$ can be reduced to the Niederreiter case:

First, the attacker uses the matrices G_i to compute the respective parity check matrices H_i, for $1 \leq i \leq N$. One way to do this is to compute the *standard form* of G_i:

$$G_i' = UG_iQ = (I_k|R),$$

where I_k is the identity matrix of size k, U an invertible matrix and Q a permutation matrix. A parity check matrix of the permuted code corresponding to G_i' is $H_i' = (-R^{\mathrm{T}}|I_{n-k})$, and a parity check matrix for G_i is $H_i = H_i'Q^{-1}$.
Next, the attacker computes the syndromes $s_i = H_i \cdot c_i^{\mathrm{T}}$. Since

$$s_i = H_i(m_1 G_i + \varphi(m_2))^{\mathrm{T}} = H_i \cdot \varphi(m_2)^{\mathrm{T}},$$

we have reduced the problem to the Niederreiter case above. Algorithm 6 outlines these steps.

Algorithm 6 Broadcast attack against the HyMES cryptosystem

INPUT: Code parameters (n, k, t), N generator matrices G_i and corresponding ciphertexts c_i, for $1 \leq i \leq N$, a finite field $\mathbb{F}_q$ and a function φ
OUTPUT: Message $m \in \mathbb{F}_q^{n+l_q}$, where $l_q = \lfloor \log_q \binom{n}{t} \rfloor$

$\forall 1 \leq i \leq N$ perform the following computations
    Find U_i and Q_i such that $G_i' = U_i G_i Q_i = (I_k|R_i)$
    $H_i' \leftarrow (-R^{\mathrm{T}}|I_{n-k})$
    $H_i \leftarrow H_i' Q_i^{-1}$
    $s_i \leftarrow H_i \cdot c_i^{\mathrm{T}}$

$H \leftarrow \langle H_1, \ldots, H_N \rangle$
$s \leftarrow (s_1| \ldots |s_N)$

Solve the linear equation system $H{m_2}'^{\mathrm{T}} = s^{\mathrm{T}}$ over $\mathbb{F}_q$ for $m_2' \in \mathcal{W}_{n,t,q}$
Solve $m_1 G_1 = c_1 - m_2'$ over $\mathbb{F}_q$ for $m_1 \in \mathbb{F}_q^n$

Return $m = (m_1|\varphi^{-1}(m_2'))$

As we noted above, the McEliece cryptosystem does not show a vulnerability to broadcast attacks as Niederreiter/HyMES do: We can sum the information contained in the

ciphertexts and public keys into a single equation by (horizontal) concatenation: $G = (G_1|\dots|G_N)$, $c = (c_1|\dots|c_N)$, $e = (e_1|\dots|e_N)$, and thus $c = mG + e$. The properties of the code generated by G are defined by the following lemma:

Lemma 3.1.2 (Concatenated codes)**.** *Let G, c, and e be defined as above. The code $\mathcal{C}$ generated by G has length nN, dimension k and minimum distance dN, the weight of e is tN.*

Proof. The length is increased to nN since every basis vector is a concatenation of N basis vectors of length n.
The dimension cannot be greater than k since this is the number of vectors generating the code $\mathcal{C}$. Also, the dimension cannot be smaller than k as this would imply that one row vector of G can be expressed as a linear combination of the remaining vectors. This is possible only if the corresponding linear combination holds for each matrix G_i, contradicting that these have a dimension of k.
Every code $\mathcal{C}_i$ generated by G_i contains a word m_i of weight d. The concatenation of these is a codeword of $\mathcal{C}$ of weight dN. If the minimum distance was smaller than dN then $\mathcal{C}$ would contain a word of weight less than dN. This word would contain, for some $i \in \mathbb{N}$, a block $[in, (i+1)n-1]$ of weight smaller than d which is a codeword of $\mathcal{C}_i$, contradicting this code's minimum distance of d. □

While the concatenated code G is less secure than the original codes G_i, this can be compensated by larger parameters, and it shows no structural weakness like in the Niederreiter case. Figure 3.1 shows the work factor to run an ISD attack against the McEliece cryptosystem using the concatenated code defined above.

We see that for an increasing number of recipients, the security of the concatenated code decreases, but it remains above a positive level. We prove the following proposition.

Proposition 3.1.3. *For an increasing number of recipients N, the code generated by $G = (G_1|\dots|G_N)$ provides a security level which is lower bounded by $C(n, R, t) = a(n) \cdot e^{Rt}$, where $a(n) \geq 1$ is a polynomial in n determined by the efficiency of the fastest ISD attack and R is the information rate of the codes $G_1, \dots, G_N$.*

Proof. As shown in [82], ISD-based algorithms have an approximate complexity of

$$C(n, R, t) = a(n) \cdot 2^{-t \cdot \log_2(1-R)},$$

where $a(n)$ is a polynomial in n. For N recipients, the security level is therefore

$$C(nN, R/N, tN).$$

Since $C(nN, R/N, tN) = a(n)\left(1 - \frac{k}{nN}\right)^{-tN}$, $\lim_{N\to\infty}\left(1 - \frac{k}{nN}\right)^{-tN} = e^{Rt}$ and $\left(1 - \frac{k}{nN}\right)^{-tN} > e^{Rt}$ for all N, the proposition follows. □

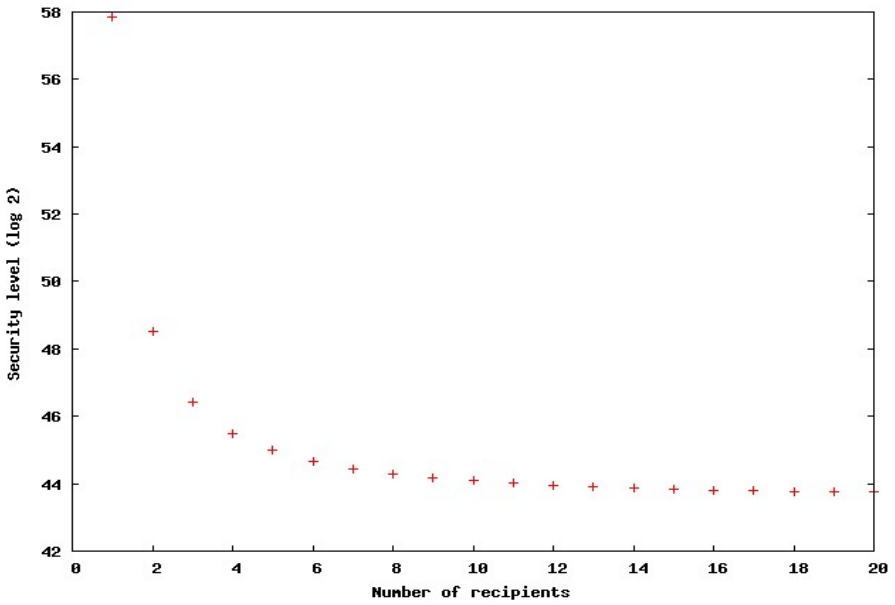


Figure 3.1.: Work factor to perform an ISD attack against the McEliece cryptosystem with parameters $(n, k, t) = (1024, 524, 50)$ using an increasing number of recipients.

3.1.3. Expected number of recipients required to break the Niederreiter/HyMES scheme

Proposition 3.1.4. *In a broadcast situation as described in Problem 3.1.1 and where the recipients use linearly independent codes of length n and co-dimension r, our broadcast attack can be run successfully if the number of recipients N_r is $N_r = \frac{n}{r}$.*

Proof. Since the codes are linearly independent, every recipient's public key adds exactly r new equations, so the system can be solved with n/r public keys. Note that in this case, r divides n and N cannot be greater than n/r; otherwise the codes could not be linearly independent. □

Proposition 3.1.5. *Let $\mathfrak{R}$ be the set of all full-rank matrices over $\mathbb{F}_q$ of size $r \times n$. In the same scenario as in Proposition 3.1.4, but with public keys chosen uniformly at random from $\mathfrak{R}$ for each user, the expected number of recipients N_r required is*

$$N_r := \left\lceil \frac{n+2}{r} \right\rceil,$$

i.e. we expect to be able to run a successful broadcast attack if $N \geq N_r$.

We start by estimating the probability that a random vector $x \in \mathbb{F}_q^n$ is linearly independent from all vectors in A, where A is a given set of r linearly independent vectors over $\mathbb{F}_q$.
A vector is linearly independent from a set of vectors if it cannot be expressed by a linear

combination of these vectors. There are q^r (different) linear combinations of vectors in A, and the whole space has dimension q^n. Thus, the probability that x is linearly independent from A is

$$\mathfrak{P} = 1 - q^{r-n} = \frac{q^n - q^r}{q^n}. \tag{3.1}$$

Therefore, if we start from the system of linear equations

$$H_1 \cdot \varphi(m)^{\mathrm{T}} = c_1^{\mathrm{T}}$$

and add the first row of H_2 to H_1, with probability $\mathfrak{P} = 1 - q^{r-n}$ we add a new equation to the system. Hence, if we assume the vectors in H_2 to be independent from each other, we expect to add $\mathfrak{P}^{-1}$ rows to get one new equation. The fact that the vectors in H_2 are not independent from each other does in fact slightly increases the chance to find a new equation: subsequent vectors in H_2 are guaranteed to be linearly independent to the previous ones already considered, so a small subset of undesired vectors is excluded. Since the effect is very small, we will continue with the probability $\mathfrak{P}$ derived above.
Thus, in order to increase the number of linearly independent equations to n, which allows us to solve the system, we need to add

$$T = \sum_{i=r}^{n-1} \frac{q^n}{q^n - q^i}$$

rows on average.

For example, for the Niederreiter parameters $[n, k] = [1024, 644]$, we need to add 646 rows, which corresponds to 3 recipients.

The expected number D of linearly dependent rows encountered when setting up the system is nearly constant: We add T rows in order to be able to solve the system, out of which $(n - r)$ are not redundant (they complement the initial r rows to a linearly independent set of n rows), and hence

$$\begin{aligned} D &= \sum_{i=r}^{n-1} \frac{q^n}{q^n - q^i} - (n - r) \\ &= \sum_{i=r}^{n-1} \left(\frac{q^n}{q^n - q^i} - 1 \right) \\ &= \sum_{i=r}^{n-1} \frac{q^i}{q^n - q^i} \\ &= \sum_{i=1}^{n-r} \frac{1}{q^i - 1} < 1.7. \end{aligned}$$

The last sum converges quickly, and it decreases with increasing q:

$$\sum_{i=1}^{n-r} \frac{1}{q^i - 1} \leq \sum_{i=1}^{\infty} \frac{1}{2^i - 1} = \sum_{i=1}^{4} \frac{1}{2^i - 1} + \sum_{i=5}^{\infty} \frac{1}{2^i - 1} \leq \sum_{i=1}^{4} \frac{1}{2^i - 1} + \sum_{i=4}^{\infty} \frac{1}{2^i} < 1.7$$

Therefore, we have

$$N_r \leq \left\lceil \frac{n+2}{r} \right\rceil .$$

In order to estimate the number of recipients N_r and thus encrypted messages we need to recover the message encrypted by the above Niederreiter/HyMES schemes, we have assumed that the parity check matrices H_i are actually random matrices of full rank even though the codes used in the two cryptosystems are not random, but Goppa codes. However, in practice Goppa codes have shown a behaviour identical to the one described above. This similarity is to be expected, since it is widely believed that the problem of distinguishing Goppa codes from random linear codes (i.e. codes defined by random generator matrices) is computationally hard (except under special circumstances [32]). Thus, the probability $\mathfrak{P}$ and the subsequent arguments above remain valid.

3.1.4. Performing a broadcast attack when $N < N_r$

When an attacker receives less than N_r broadcast messages, the resulting system $H\varphi(m)^{\mathrm{T}} = c^{\mathrm{T}}$ with $H = \langle H_1, \ldots, H_N \rangle$ is under-determined. It can be used nonetheless to mount an ISD attack. There are different ISD-like attacks, but the basic steps are as follows:

1. Choose a random $n \times n$ permutation matrix Q and compute $H' = QH$.

2. Perform Gaussian elimination on H' and c to get $(I_K|A) = c'$, where I_K is the identity matrix of size K, and $n - K$ is the number of rows of H.

3. Search for $p \leq t$ columns of A such that their sum S has Hamming distance $t - p$ to the syndrome c.

4. The non-zero entries of $S - c$ locate the remaining $t - p$ entries of $\varphi(m)$.

In a broadcast scenario where the recipients use linearly independent public keys and where $N < N_r$, the work factor of the above attack can be computed using the formulae in Proposition 4.1.3. Figure 3.2 shows the corresponding attack complexity.

In practice, the recipients' public keys might not be linearly independent. However, we expect to get only a small number of redundant equations, so the attack complexity decreases nearly identical to Figure 3.2 (compare Propositions 3.1.4 and 3.1.5).

This result is supported by [82], where Sendrier points out that ISD-based algorithms have an approximate complexity of

$$C(n, R, t) = a(n) \cdot 2^{-t \cdot \log_2(1-R)},$$

where $R = k/n$ is the information rate and $a(n)$ a polynomial in n. Increasing the number of rows in the parity check matrix decreases R, so $C(n, R, t)$ decreases exponentially.

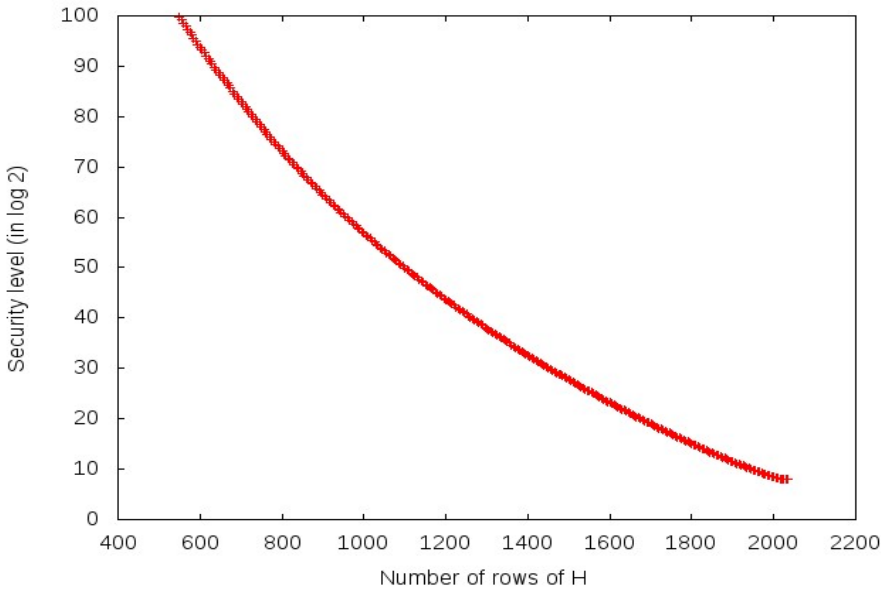


Figure 3.2.: Work factor for a broadcast attack against Niederreiter with parameters $(n, k, t) = (2048, 1498, 50)$ using ISD when $N < N_r$.

3.1.5. Related-message broadcast attack

In the previous sections, an identical message m has been sent to all recipients. In the following, we show how a broadcast attack can be performed if the messages m_i are *not* identical, but *related*. More concretely, let $M = \{m_i : 1 \leq i \leq N\}$ and we define the following property:

Definition 3.1.6. A set M of messages $m_i \in \mathbb{F}_q^n$ is called b-related if there are exactly $b \leq n$ coordinates such that all messages are identical on these coordinates. We denote this property by

$$\rho(M) = b.$$

Thus, there are $u = n - b$ bits where not all messages m_i are identical.

Proposition 3.1.7. *In a broadcast situation where the recipients use codes of length n and co-dimension r generated by linearly independent public key matrices, and where the set of messages M is b-related with $u = n - b$, the number of recipients N_r' required to solve the system of a related-message broadcast attack is*

$$N_r' = \frac{n}{r - u}.$$

Since the Niederreiter and HyMES cryptosystems do not encrypt $m_i \in M$ directly, but $\varphi(m_i)$ instead, the choice of the encoding function φ will influence $\rho(\varphi(M))$. For example, regular encoding preserves the relatedness of messages in the sense that an identical bit in m_i and m_j is equivalent to identical blocks in $\varphi(m_i)$ and $\varphi(m_j)$. This is not true for

enumerative encoding since few different bits in m_i and m_j can lead to many different bits in the encoded messages, depending on which bits differ. To keep our analysis independent of the choice of φ, we will assume that

$$\rho(\varphi(M)) = b,$$

for some value of b, and that $n - b$ is small. The attacker has to compute $b = \rho(\varphi(M))$ from his knowledge of φ and $\rho(M)$.

Solving the linear equation system

For simpler notation, let the messages in M be identical on the leftmost b bits, and (potentially) different on the rightmost $u := n - b$ bits. This can be achieved by permutation of the coordinates. Since the messages are not identical, the parity check matrices cannot be used directly to form the final system of equations. Let

$$H_i = (H_i^1 | H_i^2), \quad 1 \leq i \leq N,$$

where H_i^1 contains the leftmost b columns of H_i. We set up the following system of equations

$$\left(\begin{array}{c|cccc} H_1^1 & H_1^2 & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ H_N^1 & 0 & \cdots & 0 & H_N^2 \end{array}\right) \left(\begin{array}{c} e_0 \\ e_1 \\ \vdots \\ e_N \end{array}\right) = \left(\begin{array}{c} c_1 \\ \vdots \\ c_N \end{array}\right).$$

A solution $e = (e_0|e_1|\ldots|e_N)$ (where e_0 has length b and the other blocks have length $n - b$) yields solutions m_i to the original problem with $m_i = \varphi^{-1}(e_0|e_i)$, for $1 \leq i \leq N$.

In contrast to the identical-message broadcast attack above, every additional recipient adds equations as well as unknowns to the system. The system will eventually be solvable if and only if the number of new equations is greater than the number of new variables. This is ensured if $r > n - b = u$, and every recipient adds at least $r - u$ new equations to the system.
Therefore, the number of recipients N_r' required to solve the system of a related-message broadcast attack is

$$N_r' = \frac{n}{r - u}.$$

Similarly to Propositions 3.1.4 and 3.1.5, we modify this result for the case where the public keys are not necessarily linearly independent.

Proposition 3.1.8. *Let $\mathfrak{R}$ be the set of all full-rank matrices over $\mathbb{F}_q$ of size $r \times n$. In the same scenario as in Proposition 3.1.7, but with public keys chosen uniformly at random from $\mathfrak{R}$ for each user, the expected number of recipients N_r' required for a successful broadcast attack is*

$$N_r' = \left\lceil \frac{n + 2}{r - u} \right\rceil.$$

Table 3.2.: Runtime of our algorithm for different parameters sets for the Niederreiter encryption scheme.

Parameters (n, k, t)	Security level against ISD	Number of recipients	Runtime in sec
$(1024, 764, 26)$	57	4	6
$(1024, 524, 50)$	58	3	6
$(2048, 948, 100)$	97	2	35
$(2048, 1498, 50)$	101	4	50
$(4096, 3136, 80)$	174	5	460
$(4096, 2896, 100)$	184	4	430
$(4096, 2716, 115)$	188	3	352

Since we expect a total of at most 2 linearly dependent rows for the system (see the proof of Proposition 3.1.5), the expected number of recipients N'_r required to solve the system of a related-message broadcast attack is $N'_r = \left\lceil \frac{n+2}{r-u} \right\rceil$.

3.1.6. Implementation

We have implemented the broadcast attack described above in Java. Target was the Niederreiter cryptosystem in the FlexiProvider package [20]. Table 3.2 shows the runtime for different parameters on an Intel i5-2500 CPU using one core. Our attack is not tweaked for performance, so we expect that this time can be improved further.

Note that the runtimes increase cubic in n. This is to be expected since the main work of the attack is to solve a system of linear equations, the complexity of which is in $\mathcal{O}(n^3)$.

3.2. Countermeasures against critical attacks

Our broadcast attack exploits the fact that the received ciphertexts are related since they correspond to the same message m. A similar fact is used in other types of attacks like message-resend, related message, chosen ciphertext etc. Therefore, broadcast attacks can be prevented by using one of the CCA2 conversions that have been proposed for these other types of attacks.

3.2.1. Unsuitable conversions

Padding schemes like the well-known Optimal Asymmetric Encryption Padding (OAEP) by Bellare and Rogaway [9] are unsuitable for the McEliece, Niederreiter, and HyMES cryptosystems since they do not prevent reaction attacks: By randomly flipping bits and observing the reaction of the receiver, an attacker can recover the cleartext, and apply the inverse of the conversion to reveal the message.

There are (at least) two generic conversion, proposed by Pointcheval [78] and by Fujisaki and Okamoto [37] that work with the above cryptosystems. However, they have the

disadvantage of adding a large amount of redundancy to the ciphertexts. In both cases, the general idea is the following: Instead of encrypting the message with the (asymmetric) cryptosystem, say McEliece, it is encrypted with a symmetric system, and the corresponding key is encrypted with McEliece. Both outputs are appended to form the ciphertext. This prevents reaction attacks because the receiver will only show a positive reaction if the symmetric key can be decrypted and if it also allows to decrypt the message. Since the output block size of McEliece, Niederreiter, and HyMES is large, a lot of redundancy is thereby added which decreases the efficiency.

3.2.2. Kobara-Imai conversion

More suitable is the γ conversion by Kobara and Imai [48]. This conversion was proposed for the McEliece cryptosystem, and for large messages it manages to reduce the redundancy of the ciphertext even below that of the unconverted cryptosystem. This conversion can also be applied to the Niederreiter cryptosystem. It can not be applied to the HyMES cryptosystem, since it uses a similar technique to encode part of the message into the error vector; hence, applying the Kobara-Imai-conversion γ to the McEliece cryptosystem will achieve a similar efficiency improvement as the HyMES scheme. For the sake of completeness, we will briefly describe this conversion here. A more detailed description can be found in [13, Section 5.3]. The resulting cryptosystem is a CCA2-secure variant of Niederreiter, which allows to implement secure and efficient cryptographic applications.

This conversion consists of two modifications. Firstly, it introduces randomness into the message, thereby rendering the output indistinguishable from a random ciphertext. This prevents attacks that rely on the relation of ciphertexts and/or cleartexts, e.g. message-resend, related-message, or broadcast attacks. Secondly, both the message vector m as well as the error vector e are computed from the message. This prevents reaction attacks, since a modified error vector results in a different cleartext, which can be detected by the honest recipient.

Algorithm 7 Kobara-Imai's γ conversion, modified for the Niederreiter cryptosystem.

Notation for Algorithm 7:
$\mathbf{Enc^N}$: The Niederreiter encryption algorithm
$\mathbf{Dec^N}$: The Niederreiter decryption algorithm

Additional system parameters: The length of the random seed len_s, and a constant *const*.
Encryption Enc$^\gamma$
INPUT: Message $m \in \mathbb{F}_q^*$
OUTPUT: Ciphertext $c \in \mathbb{F}_q^w$, where $w = r + \text{len}(m) + \text{len}(c) + \text{len}(s) - l$

Generate a random seed s of length len_s
$y_1 \leftarrow \text{PRG}(s) + (m|const)$
$y_2 \leftarrow s + h(y_1)$

$(y_4|y_3) \leftarrow (y_2|y_1)$, such that
$\text{len}(y_3) = l$
$\text{len}(y_4) = \text{len}(m) + \text{len}(c) + \text{len}(s) - l$

$z \leftarrow \varphi(y_3)$
$c \leftarrow y_4|\mathbf{Enc^N}(z)$

Return c

Decryption Dec$^\gamma$
INPUT: Ciphertext $c \in \mathbb{F}_q^w$, where $w = r + \text{len}(m) + \text{len}(c) + \text{len}(s) - l$
OUTPUT: Message $m \in \mathbb{F}_q^*$

$y_4 \leftarrow \text{MSB}_{\text{len}(c)-n}(c)$
$z \leftarrow \mathbf{Dec^N}(\text{LSB}_n(c))$
$y_3 \leftarrow \varphi^{-1}(z)$
$(y_2|y_1) \leftarrow (y_4|y_3)$
$s \leftarrow y_2 + h(y_1)$
$(m|const') \leftarrow y_1 + \text{PRG}(s)$

if const'=const **then**
  Return m
else
  Reject c
end if

4. Non-Critical Attacks

Non-critical attacks have a runtime that is exponential in the security parameter (in our case, the code length, provided that the other code parameters are chosen appropriately). Therefore, they can be rendered infeasible by enlarging this parameter. Often, they are also called *decoding attacks* since many attacks in this category attempt to decode a given ciphertext, not to recover the secret key.
Because of their exponential runtime, non-critical attacks usually do not completely break the target cryptosystems. However, it is very important to understand their complexities because these determine the choice of secure parameters. We will cover the selection of parameters in more detail in Section 5. As an example, the McEliece cryptosystem itself remains secure, while the original parameters $(1024, 524, 50)$ had to be modified after Bernstein et al. [15] published an improved ISD attack with a complexity of $2^{60.4}$ binary operations against these parameters.
Code-based cryptography has an important advantage in this respect: In contrast to some other areas of post-quantum cryptography, e.g. some schemes based on lattices, it is often possible to precisely compute the expected number of computations required to run an attack against a certain cryptosystem. That means we do not have to rely on the $\mathcal{O}$-notation to measure the complexity, avoiding the hidden factors which can be very large and thus significantly influence the security. In the following, we will generally refrain from using the $\mathcal{O}$-notation; we will use it if that increases the readability of the text.

Our contributions

In this section, we present four improvements regarding non-critical attacks: A generalization of ISD attacks from binary to larger fields $\mathbb{F}_q$; an analysis of the effect of partial knowledge; an improvement of GBA attacks against certain structured matrices; and a statistical decoding algorithm generalized to $\mathbb{F}_q$, including techniques to exploit partial knowledge.
We end this section with an analysis of three "cross-area" attacks, i.e. lattice-based attacks against code-based systems and code-based attacks against lattice-based systems. Even though the attacks we analyzed were not faster than the best attacks from the respective areas, we include it here because our findings may help other researchers and lead to improvements or insights in the future.

In Section 4.1, we propose and prove lower bounds for the complexity of ISD algorithms over a finite field $\mathbb{F}_q$. We generalize the lower bounds proposed by Finiasz and Sendrier in [36] to codes over $\mathbb{F}_q$ and fix a misapproximation in their proof (the approximation of the parameter l is suboptimal). In [16], the authors pointed out that the lower bound cannot be correct by showing how to decrease the algorithm complexity below the corresponding

Table 4.1.: Examples comparing our improved ISD attack with Peters' [75].

Code parameters				Claimed	$\log_2$(#bin ops)	Lower bound
q	n	k	t	security level	(from [75])	$\log_2$(#bin ops)
256	459	204	50	80	77.10	65.05
1024	450	225	56	80	76.88	62.01
256	640	128	64	102	181.62	171.89
2	2304	1024	64	80	83.39	80.60

lower bound. We therefore propose a more general attack framework to derive the lower bound which takes the improvement in [16] into account. This improvement is based on the fact that vectors with a single non-zero entry can be added using less operations than the addition of vectors in general. While the authors of [16] also propose a new bound for the complexity of (binary) ISD, this bound is not as tight as ours. For the parameter set $(12200, 9244, 488)$ referred to in [16], their bound yields approx. 890 bits of security, compared with 991 bits for our lower bound.
In addition to that, we show how to use the structure of $\mathbb{F}_q$ to increase the algorithm efficiency by a factor of $\sqrt{q-1}$ compared to a straightforward generalization of [36]. The details of our proofs are given in Appendix A.1. Table 4.1 illustrates our improved algorithm.

Based on this generalization, in Section 4.2 we analyze two types of partial knowledge an attacker can obtain in certain scenarios:

1. The error values (of the secret error vector e) are in a subset $E \subsetneq \mathbb{F}_q$.

2. The entries of $e \in \mathbb{F}_q^n$ are known, but their positions are not.

We show that additional knowledge can be used to improve the efficiency of an attack by restricting the space that needs to be searched, and we prove new lower bounds for these cases. As an example, we analyze the Cayrel-Véron-El Yousfi (CVE) ID scheme [26] to apply our methodology and assess their method to prevent the information leak.

In the subsequent Section 4.3 we shown how the time and memory efficiency of GBA attacks against structured matrices can be improved. Our technique can be applied to a wide range of possible structures including QC and QD matrices. Even if only part of the matrix is structured, as for the truncated QC matrices in the FSB hash function, we can apply the improvement, with a correspondingly smaller effect.
However, in contrast to the claims in the corresponding paper [66], the effect can only be applied until the algorithm starts to eliminate vectors from the lists, so the effect is much smaller than originally anticipated. We will describe the technique we developed, and show why it cannot be applied once elimination takes place.

In Section 4.4, we analyze statistical decoding over a finite field $\mathbb{F}_q$. We generalize Overbeck's binary statistical decoding algorithm [72] to codes over $\mathbb{F}_q$, analyze the success probability of our algorithm, and provide experimental data for different field sizes. Our algorithm is able to successfully decode most of our test instances, even with a small sample size of $|H| = 100$. The success probability shows to be independent of the field size q, making it especially interesting for short codes over large fields $\mathbb{F}_q$. In addition to that, we describe two techniques to use knowledge about the structure of the code or about the solution to speed up the decoding algorithm.

The final Section 4.5 covers our analysis of three "cross-area" attacks:

- The sieving algorithm by Micciancio and Voulgaris [59] against code-based cryptosystems.
- Enumeration using extreme pruning by Gama et al. [41] against code-based cryptosystems.
- ISD against lattice-based cryptosystems.

We show that the pruning technique does not work in the code-context because the notion of an orthogonalized basis cannot be transferred to codes. However, the sieving and ISD algorithms can be modified to work in the corresponding area. The resulting algorithms are able to solve the hard problems of finding short codewords and short vectors, respectively, and can therefore be used to attack cryptosystems that rely on the hardness of one of these problems. As mentioned above, the attacks were not as fast as the best attack from the respective other field.

Related work

ISD over $\mathbb{F}_q$

Many improvements and generalizations of ISD-like attacks have already been proposed in the literature; we outline these pivotal examples:

1962 Prange [79]: The first proposal to use information sets for decoding.

1988 Leon [51]: Prange's proposal was made probabilistic, thereby improving its runtime.

1988 Lee-Brickell [49]: Introduced a systematic way to determine whether the recovered message is correct, and proposed several other improvements. These include bit-wise computation of vectors, which stops the computation earlier, and a speedup of the Gaussian elimination which reuses part of the columns of the parity-check matrix.

1990 van Tilburg [95]: The efficiency of the algorithm was further improved, and the ISD attack was made more efficient by a systematic method of checking and by using a random bit swapping procedure.

1989 Stern [88]: Proposed an idea that allows to make use of the birthday paradox, thereby increasing the speed of the search step.

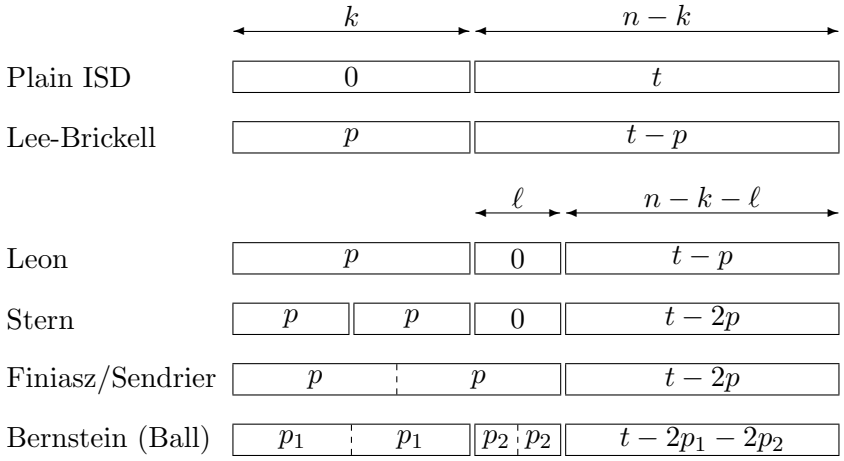
1998 Canteaut-Chabaud [22]: This paper improved the efficiency of previous algorithms and used Markov chains to model the algorithm's behavior.

2008 Bernstein et al. [15]: Several techniques were proposed to further speed up the attack algorithm, e.g. re-using pivot values to speed up the matrix inversion step.

2010 Bernstein et al. [16]: Introduced ball-collision decoding, a technique allowing to speed up the algorithm using the fact that standard basis vectors can be added very efficiently. The result is a decreased complexity of ISD, for certain parameters even below the lower bound in [36].

All above proposals are restricted to binary codes. Recently, two papers — by Finiasz and Sendrier [36] and by Peters [74] — studied ISD algorithms. The former provided lower bounds for the complexity of the ISD algorithm over $\mathbb{F}_2$, the latter described how to generalize Stern's and Lee-Brickell's algorithms to $\mathbb{F}_q$. Figure 4 illustrates the development of ISD algorithms, focussing on the error vector structure it searches for.

Figure 4.1.: Weight profile of the codewords sought by the various algorithms (Update of a figure from Overbeck and Sendrier: *Code-based Cryptography* in *Post-quantum Cryptography* [13])



Following our work, two publications contributed improvements of ISD. In their 2011 paper "Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$", May et al. [54] presented an algorithm based on Finiasz and Sendrier's ISD algorithm. The authors built on the variant with non-disjoint sets and achieved a complexity of $\tilde{\mathcal{O}}(2^{0.05363})$, slightly below that of ball-collision decoding by Bernstein et al. [16]. Since the authors state the algorithm's complexity in $\mathcal{O}()$ notation and do not determine the cut-off point with other variants of ISD (e.g. ball-collision decoding or our algorithm), however, the actual efficiency for concrete parameters is not yet known and cannot be compared with our algorithm.
The contribution of Becker et al. [2] in "Decoding Random Binary Linear Codes" is to appear at Eurocrypt 2012 and not yet available publicly.

The effect of partial knowledge

We are not aware of any previous work that specifically deals with the effects of partial knowledge.

GBA against structured matrices

The birthday paradox refers to the fact that collisions between two (usually random) lists can be found in $\mathcal{O}(2^{n/2})$, much faster than $\mathcal{O}(2^n)$ for checking every possible combination. Wagner generalized this algorithm in 2002 by allowing more than two lists. The resulting GBA takes lists $L_1, \ldots, L_t$ and an element s as input and attempts to find exactly one element $x_i \in L_i$ per list such that $\sum_i x_i = s$. By starting with t instead of 2 lists, the generalized algorithm achieves a complexity of $\mathcal{O}(2^{2\sqrt{n}})$. However, it requires a larger number of input elements and therefore more memory than the classic birthday algorithm. Minder and Sinclair's *extended k-tree algorithm* [60] can be seen as a further generalization of Wagner's work. It allows to balance the time and memory efficiency of the algorithm. This enables an attacker to optimize the attack performance in the presence of memory constraints. We will demonstrate our improvement based on Minder and Sinclair's version of the GBA.
In 2009, Bernstein et al. published a successful attack [14] against FSB_{48}, a toy version of FSB, that required just under 8 days. This attack used the two techniques mentioned in Remark 2.5.1 (page 38), but did not exploit the structure of the matrix.

Statistical decoding over $\mathbb{F}_\mathbf{q}$

Statistical decoding was introduced in 2001 by Al Jabri [45] and improved by Overbeck in 2006 [72]. We are not aware of any attempt to use statistical decoding for non-binary codes, nor to exploit knowledge about the structure of the code or of the solution to increase the algorithm's efficiency.

"Cross-area" attacks

We are not aware of any related work.

Outlook and further work

It can be seen from Table 4.2 (page 69) that the efficiency of concrete algorithms over $\mathbb{F}_2$ is not far from the lower bound, while over larger fields the gap is wider. We propose to further investigate improvements over $\mathbb{F}_q$ to decrease the size of this gap.

In addition to that, we suggest to analyze the impact of partial knowledge on other attacks, for example on GBA. Also, other types of partial knowledge should be considered. For example, in the FSB hash function [5] the "message" is transformed into a regular word, which consists of blocks of size n/t and weight 1 or, more generally, of weight t_0 for some positive integer t_0. This has been studied recently for ISD and $t_0 = 2$ by Bernstein et al. [17], and for GBA and weight k by Kirchner [46]. It should be further analyzed

how this knowledge can increase the efficiency of attacks in order to better estimate the security of cryptographic schemes.

Regarding structured codes as in Section 4.3 (page 79), we propose to analyze if and how other attacks, for example ISD, can exploit such structures. Furthermore, it would be interesting to transfer these results to other areas of cryptography, for example lattice-based schemes.
Also, we propose to analyze if the structure of the matrix can be exploited in case of a "Grover-improved GBA". In 2010, Bernstein [12] presented an improvement of an ISD attack for quantum computing. Protecting against this attack requires key sizes to be increased only by a factor of four. If structures as analyzed in our paper can also be exploited by a Grover-GBA, it would require an additional key size increase.

In the area of statistical decoding, we propose to analyze if other types of structure can be exploited as well. Especially the code structure seems to be very promising, for example if the underlying code is a Goppa code.

4.1. Information Set Decoding over $\mathbb{F}_q$

In recent years, many new proposals use codes over larger fields $\mathbb{F}_q$, mostly in an attempt to reduce the size of the public and private keys. Two examples that received a lot of attention are Berger et al.'s QC alternant [10], and Misoczki and Barreto's QD Goppa [61] codes. Cryptosystems using these codes are the QD-CFS signature scheme [8], the FSB hash function [5], the CVE ID scheme [26], and others.
Cryptographic schemes based on codes over fields other than $\mathbb{F}_2$ present security issues that are not relevant in the case of binary codes; the security of such constructions, therefore, requires separate assessment. For instance, several structural attacks against QC and QD codes have been published, indicating how to break some of the proposed non-binary parameter sets [94, 33]. These attacks use the QC/QD structure of the matrix in order to decrease the number of variables in the linear equation system which is used to recover an alternant decoder. Binary codes remain unaffected by these attacks.

ISD is one of the most important class of generic attacks against code-based cryptosystems. Attacks based on ISD are the most efficient attacks against several encryption and identification schemes, e.g. the McEliece and Niederreiter encryption schemes.

In the following, we present and prove lower bounds for ISD algorithms over a finite field $\mathbb{F}_q$. Since we focus on lower bounds in a generic attack framework instead of individual attacks, our results anticipate future software and hardware improvements and allow to compute conservative parameters for cryptographic applications.

4.1.1. Analysis of the ISD algorithm over $\mathbb{F}_q$

The security analysis of cryptosystems considers the most efficient attack and estimates its complexity. However, when a faster attack is developed, this security analysis might

become obsolete. Over the years, there have been many improvements for ISD. Finiasz and Sendrier [36] proposed to establish lower bounds for an ISD family of algorithms to allow for conservative, longer lasting security estimates. Our algorithm is based on the same framework, but includes several generalizations and improvements. This attack framework represents an idealized ISD-like algorithm, meaning that we only consider a cost for the most crucial steps and assume no cost for overhead, memory access etc. The most significant steps in ISD are the calculation of syndromes (ISD 1 and ISD 2 in Algorithm 8 on page 66), and checking the success condition (ISD 3). This approach allows us to develop lower bounds for the complexity of those attack algorithms matching our framework which is very general and includes, amongst others, the algorithms shown in Figure 4.
We start by generalizing the ISD algorithm to work over $\mathbb{F}_q$, taking the improvements from [16] into account ("Algorithm A"), and then we show how to use the field structure to increase its efficiency ("Algorithm B"). The latter algorithm is faster by a factor of $\sqrt{q-1}$ compared with the former (in number of operations, counted according to our framework).

4.1.2. A generic ISD framework

The problem we attempt to solve using ISD attacks is the GD problem. This problem can be reduced to the SD problem: Given a ciphertext $c = mG + e$, we can multiply it with a public parity-check matrix H

$$Hc^{\mathrm{T}} = H(mG + e) = He^{\mathrm{T}},$$

and solve the corresponding SD problem. A suitable matrix H can be found as described in Section 3.1.2 (page 46).

The algorithm we describe here recovers a q-ary error vector e. In each step, we randomly rearrange the columns of the parity check matrix H and apply Gaussian elimination to transform it into the form

$$H = \left(\begin{array}{c|c|c} I_{n-k-l} & 0 & H_1 \\ \hline 0 & I_l & H_2 \end{array}\right), \tag{4.1}$$

where I_{n-k-l} and I_l are identity matrices of size $(n-k-l)$ and size l, respectively. The columns are usually chosen adaptively to guarantee the success of this step, i.e. to ensure that $n-k$ linearly independent columns are found. Although this approach could bias the following steps, it has not shown any influence in practice. The variables l and p (see next step) are algorithm parameters optimized with respect to the code parameters (n, k, t).

This structure of H differs from the one proposed in [36]. We modified it in order to increase the efficiency of the syndrome generation step. This technique has been described in [16], allowing the authors to achieve an ISD complexity below the lower bound in [36]. We correct the resulting lower bounds by extending our attack framework to include this technique.

The error vector we are looking for has p_1 errors in the column set corresponding to H_1 and H_2, p_2 errors in the columns corresponding to I_l, and the remaining $(t-p_1-p_2)$ errors in the first $(n-k-l)$ columns. The reason for this split into p_1 and p_2 errors is that columns in the central block of H have exactly one non-zero entry, which makes it faster to add them to another vector. We will go into more details at the end of this section.
To find an error vector with the structure described above, many possible error patterns of $p := p_1 + p_2$ errors in the last $k+l$ columns are checked whether the weighted sum $\hat{s}$ of those p columns is identical to the syndrome s in the last l entries. This is done by searching for collisions between two sets L_1 and L_2 defined as

$$L_1 = \{H_2 e^{\mathrm{T}} : e \in W_1\}, \tag{4.2}$$
$$L_2 = \{s_2 - H_2 e^{\mathrm{T}} : e \in W_2\}, \tag{4.3}$$

where $W_1 \subseteq \mathcal{W}_{k+l;\lfloor p/2 \rfloor;q;p_1}$ and $W_2 \subseteq \mathcal{W}_{k+l;\lceil p/2 \rceil;q;p_1}$ are given to the algorithm, and $\mathcal{W}_{k+l;p;q;p_1}$ denotes the set of all q-ary words of length $k+l$ and weight p such that the first l entries contain exactly p_1 non-zero entries. This approach corresponds to the *overlapping sets* technique proposed in [36]. Alternatively, the sets can be defined such that they overlap only partially or not at all, but this has shown only a small influence in practice.
Writing $e = [e'|(e_1+e_2)]$ and $s = [s_1|s_2]$ with s_2 of length l, we thus search for vectors e_1 and e_2 of weight $\lfloor p/2 \rfloor$ and $\lceil p/2 \rceil$ respectively, such that

$$[I_l|H_2] \cdot [e_1+e_2]^{\mathrm{T}} = s_2^{\mathrm{T}}.$$

If such vectors e_1 and e_2 are found, we compute the difference $\hat{s} - s$; if it does not have weight $t-p$, the algorithm restarts. Otherwise, the non-zero entries correspond to the remaining $t-p$ errors:

$$\begin{aligned} He^{\mathrm{T}} &= \left(\begin{array}{c|c|c} I_{n-k-l} & 0 & H_1 \\ \hline 0 & I_l & H_2 \end{array}\right) \begin{pmatrix} e'^{\mathrm{T}} \\ (e_1+e_2)^{\mathrm{T}} \end{pmatrix} \\ &= \begin{pmatrix} I_{n-k-l} \cdot e'^{\mathrm{T}} + [0|H_1](e_1+e_2)^{\mathrm{T}} \\ [I_l|H_2](e_1+e_2)^{\mathrm{T}} \end{pmatrix} \\ &= \begin{pmatrix} I_{n-k-l} \cdot e'^{\mathrm{T}} \\ 0 \end{pmatrix} + \hat{s}^{\mathrm{T}} \\ &= \begin{pmatrix} s_1^{\mathrm{T}} \\ s_2^{\mathrm{T}} \end{pmatrix}. \end{aligned}$$

Therefore, we have

$$I_{n-k-l} \cdot e'^{\mathrm{T}} = s_1^{\mathrm{T}} - [0|H_1](e_1+e_2)^{\mathrm{T}},$$

revealing the remaining columns of e.

Definition 4.1.1. For any (n,k,t) code $\mathcal{C}$ over $\mathbb{F}_q$ we denote by $\mathrm{WF}_{\mathrm{qISD}}(n,r,t,q)$ the minimum work factor (in number of operations) of Algorithm 8 applied to $\mathcal{C}$, i.e. after optimizing l, p_1, p_2, W_1, and W_2 with respect to $\mathcal{C}$.

In the algorithm described in this section, all computations are done over $\mathbb{F}_q$, so the complexity also depends on the implementation of q-ary arithmetic. A naïve implementation yields an additional factor of $\log_2(q)$ for addition and $\log_2^2(q)$ for multiplication. There are several techniques to improve this, e.g. by lifting to $\mathbb{Z}[x]$ (for large q) or by precomputing exp and log tables (for small q). Especially for small q, this allows to make q-ary arithmetic nearly as fast as binary, so in order to gain conservative estimates we neglect this factor.

4.1.3. Efficiency improvement

We can use the field structure of $\mathbb{F}_q$ to increase the algorithm's efficiency. While this improvement is one of our own contributions (and has been described in [67]), it has been found independently by Minder and Sinclair [60].

Proposition 4.1.2. *The structure of $\mathbb{F}_q$ can be used to increase the efficiency of ISD (in number of operations) by a factor of $\sqrt{q-1}$ compared with a straightforward generalization of ISD to $\mathbb{F}_q$.*

To see this, note that for every vector e such that $He^{\mathrm{T}} = s^{\mathrm{T}}$, there are $q-1$ pairwise different vectors e' such that $He'^{\mathrm{T}} = as^{\mathrm{T}}$ for some $a \in \mathbb{F}_q \backslash \{0\}$, namely $e' = ae$. Clearly, if we find such an e', we can calculate e and thus solve the SD problem. We can modify the algorithm to allow it to find these vectors e' as well, thereby increasing the fraction of error vectors that are (implicitly) tested in each iteration by a factor of $q-1$ (see Appendix A.1 for a detailed description).
Since this fraction depends linearly on $|W_1| \cdot |W_2|$, we can also keep the fraction constant and decrease the size of both sets W_1 and W_2 by a factor of $\sqrt{q-1}$ each. As the work factor in each iteration of the algorithm is linear in $|W_1| + |W_2|$, this increases the algorithm's efficiency by a factor $\sqrt{q-1}$.

A simple way to decrease the size of the sets is to restrict them to vectors with the leading element equal to 1. We achieve this by redefining as follows.

$$W_1' \subseteq \{e \in \mathcal{W}_{k+l;\lfloor p/2 \rfloor;q;p_1} : \mathrm{le}(e) = 1\}, \tag{4.4}$$

$$L_1' = \left\{(H_2 e^{\mathrm{T}})(\mathrm{le}(H_2 e^{\mathrm{T}}))^{-1} : e \in W_1'\right\}, \tag{4.5}$$

$$L_2' = \left\{(s_2 - H_2 e^{\mathrm{T}})(\mathrm{le}(s_2 - H_2 e^{\mathrm{T}}))^{-1} : e \in W_2\right\}. \tag{4.6}$$

Note that in practice, the calculation of each vector is more costly due to the final division by the leading coefficient. However, this is offset by the smaller number of vectors that must be computed. Therefore, applying the above improvement to an implementation of ISD (and thus counting its number of operations instead of according to our framework) leads to an efficiency improvement smaller than a factor of $\sqrt{q-1}$.

The algorithm thus works as shown in Algorithm 8. The description is the one from [36], modified to include our improvements described above.

Algorithm 8 Information Set Decoding over $\mathbb{F}_q$ ("Algorithm B")

Notation for Algorithm 8:

$h_l(x)$	: A function returning the last l bits of the vector $x \in \mathbb{F}_q^n$
y	Notational shortcut for $\mathrm{le}(H_2 e_1^{\mathrm{T}})$
z	Notational shortcut for $\mathrm{le}(s - H_2 e_2^{\mathrm{T}})$

Parameters:

- Code parameters: integers n, $r = n - k$, and t, and a finite field $\mathbb{F}_q$.
- Algorithm parameters: two integers $p > 0$ and $l > 0$, and two sets $W_1 \subseteq \{e \in \mathcal{W}_{k+l;\lfloor p/2 \rfloor;q;p_1} : \mathrm{le}(e) = 1\}$ and $W_2 \subseteq \mathcal{W}_{k+l;\lceil p/2 \rceil;q;p_1}$.

Input: Matrix $H_0 \in \mathbb{F}_q^{r \times n}$ and a vector $s_0 \in \mathbb{F}_q^r$.

Output: A pair (P, e), consisting of permutation P and vector e, which allow to compute the message m as described above.

Repeat (MAIN LOOP)
$P \leftarrow$ random $n \times n$ permutation matrix
$(H, U) \leftarrow \mathrm{PGElim}(H_0 P)$ //Gaussian elimination as in (4.1)
$s \leftarrow s_0 U^{\mathrm{T}}$
for all $e_1 \in W_1$
  $i \leftarrow h_l(H_2 e_1^{\mathrm{T}} y^{-1})$ (ISD 1)
  $\mathrm{write}(e_1, i)$ //store e_1 in some data structure at index i
for all $e_2 \in W_2$
  $i \leftarrow h_l((s_2^{\mathrm{T}} - H_2 e_2^{\mathrm{T}}) z^{-1})$ (ISD 2)
  $S \leftarrow \mathrm{read}(i)$ //extract the elements stored at index i
  for all $e_1 \in S$
    if $\mathrm{wt}(s_2^{\mathrm{T}} - H_2(e_1 + e_2)^{\mathrm{T}}) = t - p$ (ISD 3)
      return $(P, e_1 z y^{-1} + e_2)$. (SUCCESS)

4.1.4. Lower bounds for the complexity

Proposition 4.1.3. *Let $\mathcal{C}$ be an (n,k,t) Goppa code over $\mathbb{F}_q$ and $c=(x+e)$ a McEliece ciphertext, where x is an arbitrary codeword and $e\in\mathbb{F}_q^n$ an error vector chosen uniformly at random from all vectors of weight t. If $\binom{n}{t}(q-1)^t < q^r$ (single solution), or if $\binom{n}{t}(q-1)^t \geq q^r$ (multiple solutions) and $\binom{r}{t-p}\binom{k}{p}(q-1)^t \ll q^r$, a lower bound for the expected cost (in number of operations) of using Algorithm 8 to find e is*

$$WF_{qISD}(n,r,t,q) = \min_{l;p_1;p_2} N_{p;q}(l)\cdot\left(\lambda_q^{-1}\left(\frac{2(q-1)l}{q\binom{l}{p_2'}(q-1)^{p_2'}}+p_2\right)\sqrt{\binom{k}{p_1}\binom{l}{p_2}(q-1)^{p-1}} + K_q\frac{\binom{k}{p_1}\binom{l}{p_2}(q-1)^{p-1}}{q^l}\right),$$

where

$$N_{p;q}(l)=\frac{\min\left(\binom{n}{t}(q-1)^t, q^r\right)}{\binom{r-l}{t-p}\binom{k}{p_1}\binom{l}{p_2}(q-1)^t},\quad p=p_1+p_2,\quad p_2'=\lfloor p_2/2\rfloor,$$

$$\text{and}\quad \lambda_q^{-1}=(1-\exp(-1))^{-1}\approx 1.58.$$

An exception is $p=0$, which corresponds to the classical ISD algorithm proposed by Prange [79]. In this case, we cannot gain a factor of $\sqrt{q-1}$, and the work factor is

$$WF_{qISD}(n,r,t,q)=\frac{\binom{n}{t}}{\binom{r}{t}}.$$

If $\binom{n}{t}(q-1)^t \geq q^r$ and $\binom{r}{t-p}\binom{k}{p}(q-1)^t \geq q^r$, the expected cost is

$$WF_{qISD}(n,r,t,q)\approx\min_{l;p}\frac{2lq^{r/2}}{\sqrt{\binom{r-l}{t-p}(q-1)^{t-p+1}}}.$$

Proof. We only sketch the proof here and give full details in Appendix A.1.
By computing the number of syndromes that are tested in each iteration of the algorithm and the success probability of each error pattern we try, we can compute the expected number of iterations, which corresponds to the number of executions of the steps ISD 1 and ISD 2. This number depends on the size of the sets W_1' and W_2 and we can analytically find the optimal size for these sets. In addition to that, we compute the expected number of collisions we have to test per iteration (step ISD 3 in the algorithm). Multiplied by the number of iterations, this equals the number of executions of ISD 3. Finally, we use the number of operations required to execute ISD 1-3 to compute the total cost of the algorithm. □

The variable K_q represents the number of operations required to check condition ISD 3. A realistic value for K_q is $K_q=2t$, which will be used for the parameters in Section 4.1.5. This value is justified by counting the computation of each row of $s_2^{\mathrm{T}}-H_2(e_1+e_2)^{\mathrm{T}}$ as

one operation; since every row has the same probability of yielding 0 or 1, step ISD 3 ends after $2t$ rows on average.

In [36], the cost to compute the vectors in steps (ISD1) and (ISD2) was estimated to l operations. However, we can reduce the cost by optimizing the calculation: by storing partial sums, we only have to add a single vector to compute the next syndrome. The number of operations required equals the weight of this vector, which is $l/2$ on average. In addition to that, we do a full instead of a partial Gaussian elimination, which corresponds to the central block I_l of H. These vectors contain only a single non-zero entry, which makes it particularly fast to add them in order to compute the syndrome: for every partial sum of p_1 columns, we get $\binom{l}{p_2}$ syndromes at a very low cost.

The work factor per iteration is essentially the sum of a matrix multiplication (with the permutation matrix), the Gaussian elimination, and the search for collisions between L_1' and L_2'. Compared with the binary case, the Gaussian elimination is slower in the q-ary case because every row has to be divided by the pivot entry. However, since matrix multiplication and Gaussian elimination are much faster than collision search, we allocate no cost to them.

4.1.5. Results

In [74], the author shows how to extend Lee-Brickell's and Stern's algorithms to codes over $\mathbb{F}_q$. Since our goal is to find general lower bounds, instead of the complexity of individual attacks, our assumptions are more optimistic (from an attackers point of view). In the following, we show how much each of our assumptions contributes to the difference between our results on those in [74].
The website [75] lists the work factor of the latter algorithm against several parameters. In Tables 4.2 and 4.3, we use the same parameters and compare these results with our lower bound.

As noted above, the complexity of the binary ISD algorithm is relatively close to our lower bound, while the gap is wider in the q-ary case. This is due to our efficiency improvement of a factor of $\sqrt{q-1}$ and the fact that over $\mathbb{F}_q$ the cost of the individual steps of current ISD algorithms is not as close to our assumptions (e.g. no cost for Gaussian elimination).

4.1.6. Comparison with other results on the ISD complexity

For the algorithm from [74], as well as for our lower bound algorithm, the expected number of operations is the product of the number of iterations by the number of operations in each iteration. While the former factor is the same for both algorithms, or even a little higher for our algorithm, the lower bound for the number of operations per iteration is much smaller in our case, which results in the difference between these algorithms.
The following comparison is between our algorithm and the overlapping-sets variant in [74] which is structurally closer to our algorithm than the one using even-split sets. The run-

Table 4.2.: Parameters analyzed by Peters [75]. The claimed security is taken from [10] and [61], the last two columns show Peters' complexity and our lower bound.

Code parameters				Claimed	$\log_2$(#bin ops)	Lower bound
q	n	r	t	security level	(from [75])	$\log_2$(#bin ops)
256	459	204	50	80	77.10	65.05
256	510	204	50	90	84.87	72.94
256	612	204	50	100	98.29	86.50
256	765	255	50	120	97.52	85.14
1024	450	225	56	80	76.88	62.01
1024	558	279	63	90	84.00	69.17
1024	744	372	54	110	70.54	57.88
4	2560	1024	128	128	187.46	178.85
16	1408	512	128	128	210.76	204.65
256	640	128	64	102	181.62	171.89
256	768	256	128	136	253.01	243.02
256	1024	512	256	168	329.04	318.65
2	2304	1024	64	80	83.39	80.60
2	3584	2048	128	112	112.18	109.98
2	4096	2048	128	128	136.47	134.51
2	7168	4096	256	192	215.91	213.92
2	8192	4096	256	256	265.01	262.42

Table 4.3.: The two parameter sets from [16], which are below the lower bound proposed in [36] (number of bit operations in $\log_2$)

Code parameters				Lower bound	#bin ops	Our new
q	n	k	t	from [36]	(from [16])	lower bound
2	6624	5129	117	255.18	254.15	251.95
2	12200	9244	488	999.45	996.22	991.94

time difference between these two variants is comparatively low.

The number of operations per iteration for the first algorithm is the sum of three steps:

1. Reusing parts of information sets and performing precomputations,
2. Compute sums of p rows on l bits to calculate $H_2 e^{\mathrm{T}}$,
3. For each collision (e_1, e_2), checking if $\mathrm{wt}(s_2^{\mathrm{T}} - H_2(e_1 + e_2)^{\mathrm{T}}) = t - p$.

To compare the cost of these steps with that used for our lower bound, we calculate all values for the $(450, 225, 56)$ parameter set over $\mathbb{F}_{1024}$. For this set, using $p = 1$, $l = 3$, $m = 1$, $c = 2$, and $r' = 1$ (the last variable is called r in [74], but it should not be confused with the co-dimension), we calculate a total cost of the first algorithm of $2^{72.47}$, which consists of $2^{51.46}$ iterations of $2^{21.0}$ operations[1].

1. Precomputations

The cost of the first step is given in [74] as

$$(n-1)\left((k-1)\left(1-\frac{1}{q^{r'}}\right)+(q^{r'}-r')\right)\frac{c}{r'},$$

where c and r' are algorithm parameters. For these parameters, this amounts to $2^{20.1}$ operations, and it is the most expensive step.
Our algorithm does not use precomputation, so we allocate no cost.

2. Computing sums of p rows to calculate $H_2 e^{\mathbf{T}}$

The cost of this step for the first algorithm is

$$((k-p+1)+(|W_1|+|W_2|)(q-1)^p)\,l.$$

For the parameters given above, this step adds $2^{19.9}$ operations.

Our algorithm allocates to this step a cost of

$$\begin{aligned}&\frac{(q-1)l|W_1'|}{q\binom{l}{p_2'}(q-1)^{p_2'}}+|W_1'|p_2'+\frac{(q-1)l|W_2|}{q\binom{l}{p_2'}(q-1)^{p_2'}}+|W_2|p_2'\\ =\;&\left(\frac{2(q-1)l}{q\binom{l}{p_2'}(q-1)^{p_2'}}+p_2\right)\sqrt{\binom{k}{p_1}\binom{l}{p_2}(q-1)^{p-1}}.\end{aligned}$$

We make this optimistic assumption[2] for the cost of a matrix-vector multiplication to anticipate further software and hardware improvements for this operation. The result is $2^{5.6}$ operations in this case.

[1] Computed with the formulae in [74]. We are unable to determine why the authors website [75] claims a different complexity.
[2] From the cryptanalyst's point of view.

Table 4.4.: Contribution of individual steps to the complexity difference.

Step	Peters' complexity	Our lower bound
1. Precomputations	$2^{20.1}$	0
2. Computing sums	$2^{19.9}$	$2^{5.6}$
3. Checking collisions	$2^{12.4}$	$2^{4.6}$
Total	$2^{21.0}$	$2^{6.2}$

3. Checking collisions

The first algorithm allocates a cost of

$$\frac{q}{q-1}(t-2p)2p\left(1+\frac{q-2}{q-1}\right)\frac{|W_1|\cdot|W_2|(q-1)^{2p}}{q^l}$$

to this step. For our set of parameters, this equals $2^{12.4}$ operations.

In our algorithm, we expect the number of collisions to be

$$\frac{\lambda_q|W_1'|\cdot|W_2|}{q^l}=\frac{\binom{k}{p_1}\binom{l}{p_2}(q-1)^{p-1}}{q^l}.$$

The cost K_q of checking each collision is taken to be $K_q = 2t$. Thus, the expected cost per iteration is $2^{4.6}$.

Some of the assumptions above may seem fairly optimistic. However, we find it necessary to make these assumptions since we want to establish conservative lower bounds.

4.1.7. Conclusion

For the three steps above, we have the following comparison:

In total, the work per iteration of our algorithm is smaller by a factor of $2^{14.8}$. This number reflects the assumptions we made to establish lower bounds for the complexity. While our assumptions lead to a higher number of iterations — $2^{55.8}$ compared with $2^{51.5}$ — the total number of operations is smaller by a factor of $2^{10.4}$ in our case.

4.2. ISD using partial knowledge

While most security proofs assume that an attacker does not have any additional information about the secret key, we show that in certain scenarios an attacker can gain partial knowledge of the secret. In this section, we present how this knowledge can be used to improve the efficiency of an ISD attack. The modified attack algorithms are at least as efficient as the unmodified version, but usually they are significantly better. For most

types of partial knowledge discussed in the following sections we use the framework from Section 4.1.1 to derive new bounds for the complexity of such an attack; in one case we present arguments to show that the modified attack is more efficient and to allow the approximation of the improvement factor.
We analyze two types of partial knowledge, present applications for our results, and give an idea how to prevent the leakage of such knowledge to an attacker. These two types are:

1. Error values are in a subset $E \subsetneq \mathbb{F}_q$, i.e. the error vector $e \in E_0^n \subsetneq \mathbb{F}_q^n$ with $E_0 = E \cup \{0\}$.

2. The entries of $e \in \mathbb{F}_q^n$ are known, but their positions are not.

Note that these types of partial knowledge only apply over fields $\mathbb{F}_q$ larger than binary; the first type is not possible in $\mathbb{F}_2$, and the second type is trivial (no additional information).

There are various situations when adversaries have partial knowledge. For example, any Stern-like identification scheme that works over $\mathbb{F}_q$, provided the scheme does not use measures to prevent it, leaks information of the second type we described above: For one of the challenges sent by the verifier, the prover sends a random permutation of the secret key in the answer phase. This reveals the value of all entries of the secret, but not their positions.
There are other types of partial knowledge, e.g. knowledge about the structure of the underlying code. Also, other attacks (e.g. GBA) should be able to exploit partial knowledge. We leave these research questions for future work.

As an example, we show how to apply partial knowledge to attack the CVE ID scheme if information leakage is not prevented. In this case, the complexity of the attack is decreased significantly. Clearly, the potential leakage of partial knowledge has to be analyzed when designing new cryptosystems.

4.2.1. Error values are in a set $E \subsetneq \mathbb{F}_q$

If we know that the error values are in a proper subset $E \subsetneq \mathbb{F}_q$, we can limit the size of the sets W_i and L_i by redefining:

$$W_1 \subseteq E_0^{k+l} \cap \{e \in \mathcal{W}_{k+l;\lceil p/2 \rceil;q;p_1} : \mathrm{le}(e) = 1\}, \tag{4.7}$$

$$W_2 \subseteq E_0^{k+l} \cap \{e \in \mathcal{W}_{k+l;\lfloor p/2 \rfloor;q;p_1}\}, \tag{4.8}$$

$$L_1 = \left\{(H_2 e^{\mathrm{T}})(\mathrm{le}(H_2 e^{\mathrm{T}}))^{-1} : e \in W_1\right\}, \tag{4.9}$$

$$L_2 = \left\{(s_2 - H_2 e^{\mathrm{T}})(\mathrm{le}(s_2 - H_2 e^{\mathrm{T}}))^{-1} : e \in W_2\right\}, \tag{4.10}$$

where $E_0 = E \cup \{0\}$. When restricting the leading entry of e to 1, the above assumes that $1 \in E$. If this is not the case, i.e. $1 \notin E$, any other element in E can be used to fix the leading entry of all vectors in W_1. This gives the following proposition:

Proposition 4.2.1 (Error values in a set E)**.** *Let $\mathcal{C}$ be an (n,k,t) Goppa code over $\mathbb{F}_q$ and $c = x + e$ a McEliece ciphertext, where x is an arbitrary codeword and $e \in E_0^n$ an error vector chosen uniformly at random from all vectors of weight t. If $\binom{n}{t}|E|^t < q^r$ (single solution), or if $\binom{n}{t}|E|^t \geq q^r$ (multiple solutions) and $\binom{r}{t-p}\binom{k}{p}|E|^t \ll q^r$, a lower bound for the expected cost (in number of operations) of using Algorithm 8 and the improvement technique above to find e is*

$$\begin{aligned} & WF_{qISD}(n,r,t,q) \\ &= \min_{l;p_1;p_2} N_{p;q}(l) \cdot \left(\lambda_q^{-1} \left(\frac{2|E|l}{(|E|+1)\binom{l}{p_2'}(q-1)^{p_2'}} + p_2 \right) \sqrt{\binom{k}{p_1}\binom{l}{p_2}|E|^{p-1}} \right. \\ &+ \left. K_q \frac{\binom{k}{p_1}\binom{l}{p_2}|E|^{p-1}}{q^l} \right), \end{aligned}$$

where

$$N_{p;q}(l) = \frac{\min\left(\binom{n}{t}|E|^t, q^r\right)}{\binom{r-l}{t-p}\binom{k}{p_1}\binom{l}{p_2}|E|^t}, \quad p = p_1 + p_2, \quad p_2' = \lfloor p_2/2 \rfloor,$$
$$\text{and} \quad \lambda_q^{-1} = (1 - \exp(-1))^{-1} \approx 1.58.$$

The case $p = 0$ is identical to the one in Proposition 4.1.3, hence

$$WF_{qISD}(n,r,t,q) = \frac{\binom{n}{t}}{\binom{r}{t}}.$$

If $\binom{n}{t}|E|^t \geq q^r$ and $\binom{r}{t-p}\binom{k}{p}|E|^t \geq q^r$, the expected cost is

$$WF_{qISD}(n,r,t,q) \approx \min_{l;p} \frac{2lq^{r/2}}{\sqrt{\binom{r-l}{t-p}|E|^{t-p+1}}}.$$

Proof. See Appendix A.2.
The difference to the proof of Proposition 4.1.3 is that we can decrease the size of the sets W_1' and W_2 by allowing only error patterns with values in E_0. The expected number of iterations is unchanged, but the number of operations per iteration decreases rapidly with E. □

4.2.2. Error values known (but not error positions)

Let the error values be $v_1, \ldots, v_t \in \mathbb{F}_q$. Depending on the size of the set $V := \{v_1, \ldots, v_t\}$, several strategies can be used. We will describe three cases:

1. $|V| = t$, i.e. all error values are different.
2. $|V| < t$, but $|V| \gg 1$
3. Most error values belong to a very small subset of V (or $|V| \ll t$).

1. $|\mathbf{V}| = \mathbf{t}$

In this case, we can decrease the number of error vectors we have to try in each iteration in the following way. For each distribution of $p = p_1 + p_2$ errors in $k + l$ locations, we do not assign all combinations of q-ary values to the error positions, but instead randomly choose p out of the t known error values and assign them in all possible combinations. Hence, instead of

$$\binom{k}{p_1}\binom{l}{p_2}(q-1)^{p-1}$$

combinations, we only have to test

$$\binom{k}{p_1}\binom{l}{p_2}\binom{t}{p}p!.$$

For most parameters, the second expression is much smaller than the first. For these parameters, we redefine the sets W_i and L_i as follows

$$\begin{aligned}
W_1 &\subseteq \{e \in \mathcal{W}'_{k+l;\lceil p/2\rceil;q;p_1}\}, && (4.11)\\
W_2 &\subseteq \{e \in \mathcal{W}'_{k+l;\lfloor p/2\rfloor;q;p_1}\}, && (4.12)\\
L_1 &= \left\{(H_2e^{\mathrm{T}})(\mathrm{le}(H_2e^{\mathrm{T}}))^{-1} : e \in W_1\right\}, && (4.13)\\
L_2 &= \left\{(s_2 - H_2e^{\mathrm{T}})(\mathrm{le}(s_2 - H_2e^{\mathrm{T}}))^{-1} : e \in W_2\right\}, && (4.14)
\end{aligned}$$

where the p error values in each element of $\mathcal{W}'_{k+l;\lceil p/2\rceil;q;p_1}$ are taken from the known t error values. This gives the following result:

Proposition 4.2.2 (Known error values)**.** *Let $\mathcal{C}$ be an (n, k, t) Goppa code over $\mathbb{F}_q$ and $c = x+e$ a McEliece ciphertext, where x is an arbitrary codeword and $e \in \mathbb{F}_q^n$ an error vector chosen uniformly at random from all vectors of weight t. Let $|V| = t$ as described above. If $\binom{n}{t}t! < q^r$ (single solution), or if $\binom{n}{t}t! \geq q^r$ (multiple solutions) and $\binom{r}{t-p}\binom{k}{p}p!(t-p)! \ll q^r$, a lower bound for the expected cost (in number of operations) of using Algorithm 8 and the improvement technique above to find e is*

$$\begin{aligned}
WF_{qISD}(n,r,t,q) &= \min_{l;p_1;p_2} N_{p;q}(l)\cdot\left(\lambda_q^{-1}\left(\frac{2(q-1)l}{q\binom{l}{p'_2}\binom{t}{p'_2}p'_2!}+p_2\right)\sqrt{\binom{k}{p_1}\binom{l}{p_2}\binom{t}{p}p!}\right.\\
&\qquad \left. + K_q\frac{\binom{k}{p_1}\binom{l}{p_2}\binom{t}{p}p!}{q^l}\right),
\end{aligned}$$

where

$$\begin{aligned}
N_{p;q}(l) &= \frac{\min(\binom{n}{t}t!, q^r)}{\binom{r-l}{t-p}\binom{k}{p_1}\binom{l}{p_2}\binom{t}{p}p!(t-p)!}, \quad p = p_1 + p_2, \quad p'_2 = \lfloor p_2/2\rfloor, \quad \textit{and}\\
\lambda_q^{-1} &= (1-\exp(-1))^{-1} \approx 1.58.
\end{aligned}$$

An exception is $p = 0$ where the above method does not gain anything, hence

$$WF_{qISD}(n, r, t, q) = \frac{\binom{n}{t}}{\binom{r}{t}}.$$

If $\binom{n}{t}t! \geq q^r$ and $\binom{r}{t-p}\binom{k}{p}p!(t-p)! \geq q^r$, the expected cost is

$$WF_{qISD}(n, r, t, q) \approx \min_{l;p} \frac{2lq^{r/2}}{\sqrt{\binom{r-l}{t-p}(t-p)!}}.$$

Proof. See Appendix A.3.
Again, this type of partial knowledge allows to decrease the size of the sets W_1' and W_2. The method to construct these sets is different compared to the case above. Instead of restricting the allowed error values, we redefine the sets W_1' and W_2 such that the entries have the correct weight and weight distribution, and in addition that the entries are chosen from v_1 to v_t. This last condition explains the additional binomial factors in the resulting formula of the lower bound. □

2. $|\mathbf{V}| < \mathbf{t}$, and many different error values

This case enables further improvement compared to the previous Section 4.2.2. Because of the condition $|V| < t$, some error values occur more than once. This further decreases the size of the set $\mathcal{W}'_{k+l;p;q;p_1}$, and hence W_i and L_i decrease in size.
For each of the $\binom{k+l}{p}$ patterns to distribute the errors, p values are chosen from V to be allocated to these error positions. If it happens that the same value is drawn more than once, then some permutations of these p error values lead to the same result, and we only need to consider one of these permutations.

The exact efficiency improvement depends not only on $|V|$, but on the detailed distribution of errors (e.g., three times the same error value is not the same as two pairs of the same value each). Let D_i, $i \in \mathbb{N}$, describe the distributions of error values that can occur after applying the random permutation, i.e. every D_i represents a certain number of pairs, triplets etc. Let $P(D_i)$, $i \in \mathbb{N}$, be the probability that such a distribution occurs, and $G(D_i)$ be the efficiency gain in such a case (with $0 < G(D_i) \leq 1$). Then

$$\sum_{i \in \mathbb{N}} P(D_i) = 1,$$

and the total work factor is decreased by a factor of

$$\sum_{i \in \mathbb{N}} P(D_i) \cdot G(D_i).$$

As an example, consider a case where $t = 20$, $p = 6$, $|V| = 10$, and $v_i = \lceil i/2 \rceil$; that is, all error values come in pairs. We can describe each distribution by the number j of error

Table 4.5.: Expected efficiency gain for $t = 12$ and various values of p and v. Shown is the fraction of required operations between the improved and the standard algorithm.

	$p = 2$	$p = 3$	$p = 4$	$p = 6$
$v = 4$	0.91	0.74	0.53	0.19
$v = 3$	0.86	0.63	0.38	0.09
$v = 2$	0.72	0.44	0.12	0.02

value pairs amongst the p last error positions (where collisions are searched for). Each pair decreases the number of operations by a factor of 2. The probability of the event is

$$P(j) = \frac{\binom{t/2}{j}\binom{t/2-j}{p-2j}2^{t/2-j}}{\binom{t}{p}}.$$

Hence, the number of operations performed by the algorithm is reduced to $\sum_{j=0}^{3} P(j) \cdot 2^{-j}$, an improvement of $\approx 36\%$.

In this example, we do not have a huge efficiency improvement, but for smaller V (and hence more pairs, triplets etc. of error values), the chance of a gain as well as the gain itself increases.

In order to analyze the situation in more detail, we introduce some notation: Let $v = |V|$ denote the size of V; each of these v values occurs at $u_1, \ldots, u_v$ positions, and a configuration $(c_1, \ldots, c_v)$ with $\sum_{i=1}^{v} c_i = p$ denotes how many of these values are chosen for the last p error positions. Then the probability of a certain configuration $c = (c_1, \ldots, c_v)$ is

$$P(c) = \frac{\prod_{i=1}^{v}\binom{u_i}{c_i}}{\binom{t}{p}},$$

while the gain is

$$G(c) = \prod_{j=1}^{v}(c_i)!.$$

An interesting general case is $u_1 = \ldots = u_v = t/v$. Since p is usually small for ISD attacks, we do not expect a large efficiency gain if v is large as well. However, for small values of v, the gain increases significantly. See Table 4.5 for details.

Note that higher values of p allow for greater efficiency improvements. This has to be taken into account when selecting attack parameters for the ISD algorithm.

3. Most error values belonging to a very small subset of V (or $|\mathbf{V}| \ll \mathbf{t}$)

If most error values come from a smaller subset $V_s \subset V$, we can modify the algorithm to take advantage of this fact. Instead of considering the full set V when defining sets L_1

and L_2 and searching for collisions, we can simply assume that the random permutation moved all errors with values not belonging to V_s to the left hand side of the matrix. The sets L_1 and L_2 are then defined using the smaller set V_s, decreasing their size and thereby increasing the algorithm's efficiency. While this assumption decreases the probability of finding a suitable permutation, it can be more than offset by the reduced number of operations per iteration.
The case $|V| \ll t$ is a special case with $V_s = V$.

4.2.3. Example: The CVE ID scheme

In [26], the authors propose a zero-knowledge identification scheme called CVE based on the q-ary SD problem. They propose to use a special permutation of the secret value s in this scheme in order to hide the non-zero values of s. In this section, we study the effect of a leakage of the values of s on the ISD algorithm.
We first introduce the special permutation that is used in the protocol to hide the non-zero values.

Definition 4.2.3. Let γ be a permutation of $\{1, \ldots, n\}$ and $w = (w_1, \ldots, w_n) \in \mathbb{F}_q^n$ such that $\forall i, w_i \neq 0$. We define the transformation $\Pi_{w,\gamma}$ as :

$$\begin{array}{rccc} \Pi_{w,\gamma}: & \mathbb{F}_q^n & \longrightarrow & \mathbb{F}_q^n \\ & v & \mapsto & (w_{\gamma(1)}v_{\gamma(1)}, \ldots, w_{\gamma(n)}v_{\gamma(n)}) \end{array}$$

Notice that $\forall \alpha \in \mathbb{F}_q$, $\forall v \in \mathbb{F}_q^n$, it holds that $\Pi_{w,\gamma}(\alpha v) = \alpha \Pi_{w,\gamma}(v)$, and $\mathrm{wt}(\Pi_{w,\gamma}(v)) = \mathrm{wt}(v)$.

Attack scenario

In this section, we show how an ISD attacker against the security of the CVE can gain partial knowledge of the secret. We analyze the CVE scheme and show that the prover leaks information to a potential attacker when using a "classic" permutation instead of the function $\Pi_{w,\gamma}$.

We assume that the attacker Eve attempts to impersonate the honest prover Alice. She can do this by recovering Alice's secret e; when Eve later runs the ID scheme with the honest verifier Bob, this allows her to successfully answer all queries.

First, Eve runs the ID scheme with Alice, where Eve sends the challenge $b = 1$. If the CVE scheme does not use the function $\Pi_{w,\gamma}$ to prevent information leakage, but rather sends the permuted secret, Eve will gain partial knowledge on the secret e.
Similarly to the original Stern scheme, Eve then attempts to find a codeword w, minimizing the weight of $(v + w)$ where $v \in H^{-1}(s)$ and s is Alice's public key. If she is successful, Eve can use $s' := v + w$ to successfully answer all queries by the verifier Bob. An element w with the above properties can be found using ISD-like algorithms.

Table 4.6 shows how much our modification of ISD attacks can decrease the security of the CVE scheme (without the special permutation). We will give the ISD complexity for three cases (as seen from an attackers point of view):

Table 4.6.: Parameter sets for the CVE ID scheme (without the special permutation) over $\mathbb{F}_q$. (P, T) refers to the number of pairs and triples we assume for this case. The field size is $q = 256$ for all parameters.

	Security	Security level modified ISD and (P, T)					
(n, k, t)	level ISD	Pessimistic		Typical		Optimistic	
$[128, 64, 49]$	71.93	76.06	$(0, 0)$	72.06	$(4, 0)$	67.48	$(6, 1)$
$[144, 72, 55]$	80.76	85.03	$(0, 0)$	80.03	$(5, 0)$	75.45	$(7, 1)$
$[208, 104, 78]$	113.78	118.39	$(0, 0)$	106.81	$(9, 1)$	101.22	$(12, 2)$
$[256, 128, 97]$	142.61	147.12	$(0, 0)$	129.95	$(12, 2)$	124.37	$(15, 3)$

- Pessimistic case: All error values are different (i.e. the least improvement potential)
- Typical case: The expected number of pairs, triples etc.
- Optimistic case: More than the expected number of pairs, triples etc.

Note that in the pessimistic case (and in one of the typical cases), the complexity of the modified algorithm is higher compared with the unmodified version. This is due to the fact that the optimal value of p is very small, so the modification offers only a very small advantage, which is more than offset by the disadvantage of not including the improvement factor of $\sqrt{q-1}$ described in Section 4.1.3. The other cases, however, allow for a significant improvement compared with the unmodified version.

Preventing the information leakage

The special permutation $\Pi_{w,\gamma}$ is introduced in [26] and used instead of a normal permutation. The vector w hides the entries of s, thereby preventing the attack described above. Since in some cases the prover has to send w to the verifier, so this modification increases the communication cost; for the parameters in [26], this corresponds to an increase from 31 kbit to 33 kbit for a security level of 80. However, simply increasing the code parameters (n, k, t) to offset the loss in security would be far more costly.

4.2.4. Implications of partial knowledge attacks on the parameters

If the information leakage can not be prevented, the choice of parameters has to be modified in order to offset the reduction in security. This is done in three steps:

1. Determine the required security level using the methodology introduced in Section 5.1.
2. Estimate the loss in security as described above and add it to the security level.
3. Find optimal parameters in the row of Table 5.2 corresponding to this new security level.

4.3. Improved GBA attacks against structured matrices

As pointed out in Section 2.3.3, there have been several proposals to use codes with additional structure in order to reduce the public key size. Most recently, QC alternant [10] and QD Goppa [61] codes, the QD-CFS signature scheme [8], and the FSB hash function [5]. In this section, we show how a broad class of such structures can be exploited to increase the time and memory efficiency of a GBA attack, which is one of the best generic attacks against code-based cryptosystems. The GBA attack has been proposed by Wagner in 2002 [97] and was generalized by Minder and Sinclair in 2009 [60]. It is the best attack against several cryptosystems.
However, our results in [66] were erroneous and the true effect is much smaller than assumed. Our technique works, but it can only be applied until the algorithm starts to eliminate vectors from the lists; after that, the time and space complexity is identical to the unmodified GBA: The number of computations and memory access of our modified GBA algorithm is decreased by a factor of up to r compared with the unmodified algorithm. For parameters suitable for cryptography, the improvement factor cannot be much larger than 2, since these parameters are designed such that elimination of vectors is necessary.
We will describe the technique we developed, and show why it cannot be applied once elimination takes place.

4.3.1. Efficiency improvement

Preliminaries

Our analysis is applicable in all cases where the underlying code has a parity check matrix of the following structure: Each row of the parity check matrix H is a permutation of the first row. This is true, for example, for the cryptosystems based on QC and QD codes in [10, 61, 8].
This means that we have permutations Θ_i, $1 \leq i \leq r$, such that the i-th row H_i of H can be computed by

$$H_i = \Theta_i(H_1) \quad \text{for } 1 \leq i \leq r.$$

We abuse notation and write, for integers a and b, $\Theta_i(a) = b$, to denote that Θ_i permutes the a-th entry to position b.

Definition 4.3.1. To efficiently address the list entries, we note that for every entry x in the lists there are unique $c_1 < c_2 < \ldots < c_k$ and $h_1, h_2, \ldots, h_k$, such that x was created by the weighted sum of columns of H:

$$x = \sum_{i=1}^{k} c_i H_{\cdot,h_k}.$$

We define $\text{index}(x) := (c_1, \ldots, c_k; h_1, \ldots, h_k)$ and write

$$\Theta_i((c_1, \ldots, c_k; h_1, \ldots, h_k)) = (c_1, \ldots, c_k; \Theta_i(h_1), \ldots, \Theta_i(h_k)). \tag{4.15}$$

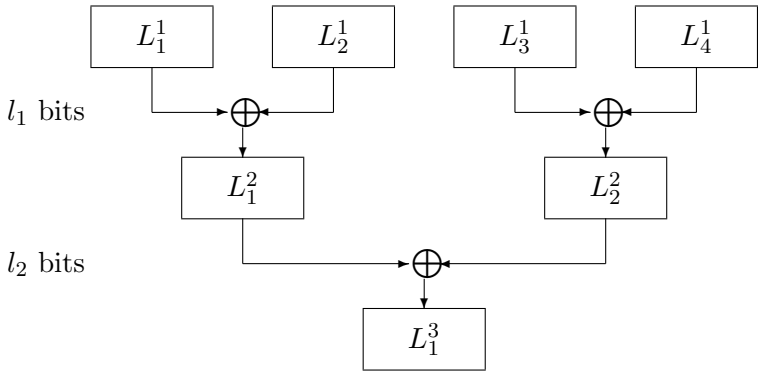
Since an index uniquely defines an element, we can associate the two and write for short $y := \text{index}(x)$. For any vector x, let us denote its i-th entry by $x[i]$. Then

$$x[i] = \Theta_i^{-1}(\text{index}(x))[1]. \tag{4.16}$$

Improved algorithm

We will describe our improved algorithm as a modification of the one by Minder and Sinclair [60]; Figure 4.2 illustrates the basic GBA.

Figure 4.2.: Illustration of the GBA for $k = 4$.



We start with $t = 2^a$ lists $L_1^1, \ldots, L_t^1$. In our case, each lists contains n entries, the columns of H. However, due to the structure of H, we only need to store the first bit of each entry, corresponding to the first row of H. We let l_i denote the number of bits that is forced to zero in the i-th merge step.
This setup assumes $s = 0$. If $s \neq 0$, we modify the list L_t^1 in a similar way as described in Section 2.5.2, by subtracting s from every entry of L_t^1. We will later go into more details.
In the first merge step, we calculate the first bit of the entries in L_i^2 by adding pairs of entries of L_{2i-1}^1 and L_{2i}^1. If $l_1 = 0$ we continue with the second merge step — again, only calculating the first bit of L_i^3, and so on.

The elimination issue

As soon as one of the l_j is greater than zero, we cannot use our technique anymore. The reason is that in the next merge step (following the elimination), most list entries $\Theta_i^{-1}(\text{index}(x))$ cannot be found since they have been eliminated. Also, in general we cannot conclude from this that $\Theta_i^{-1}(\text{index}(x))[1] \neq 0$ because the list entry might have been eliminated in any of the previous merge and elimination steps.
In order to eliminate all vectors from L_i^2 whose first l_i bits contain non-zero values, we need to compute the remaining bits of each list entry using equation (4.16). For the remaining steps of the algorithm, these bits need to be stored to enable us to continue with the GBA. Therefore, in each merge step j where we can apply our technique, we have to compute and store only the first bit of each vector of the current lists L_i^j, increasing time and space

efficiency by a factor of up to r. The remaining merge steps are identical to the unmodified GBA algorithm, and the total space and time efficiency improvement is between a factor of 1 and r, depending on the number of steps with and without elimination and the corresponding list sizes.
The pseudo code of our algorithm is shown in Algorithm 10. The operand $\oplus_1$ refers to the operation of creating the new list from sums of all pairs of the two parent lists, only computing and storing the first bit, as shown in Algorithm 9 (here, | refers to the concatenation operation):

Algorithm 9 Pseudo code for $C := A \oplus_q B$

1: **for all** $a \in A, b \in B$ **do**
2: $\quad C \leftarrow C|(a[1] + b[1] \mod q)$
3: **end for**

Let $\oplus$ denote the corresponding merge operation where all bits are computed.

The GBA algorithm described above assumes $s = 0$. If $s \neq 0$, we can modify it similarly as Wagner proposed in [97]: The rightmost list L_t^1 does not get columns $(H_{\cdot,i})$ of H, but $(H_{\cdot,i} - s)$. This allows us to search for list elements that sum to zero, and the corresponding columns of H will sum to s.
For the merge step of each rightmost list, we have to modify equation (4.16), since different rows use different bits of s as an offset:

$$x[i] = \Theta_i^{-1}(\text{index}(x))[1] + s[1] - s[i]. \tag{4.17}$$

In practice, our modified GBA can be implemented efficiently by using look-up tables for the Θ_i, and by using a suitable data structure for the lists L_i^j.

Analysis of time and memory efficiency

Our modification to the GBA reduces the number of computations and memory operations. The memory required *before* the first elimination is reduced by the same factor. However, since the lists have to be expanded once elimination takes place, the total memory required is unchanged in most applications. There can be an effect, though, if partial results — lists where no vectors have been eliminated — are stored temporarily to continue computation at a later point. This was done, for example, in the attack on the toy version of FSB [14].
As we described above, our modified GBA algorithm decreases the number of computations and memory access by a factor of up to r. For parameters suitable for cryptography, though, the effect is small. These parameters lead to very large list sizes, since this ensures the necessary level of security, and thus elimination is required to prevent the list from growing to a size that can not be handled. Since elimination starts when the list sizes get large, and since our improvement cannot be applied in the elimination phase, those steps with the highest cost are not improved. In order to estimate the maximum improvement factor in this case, we approximate the complexity of a merge step of the GBA by the size of the child list, as proposed in [60]. In the same paper, the authors prove that the size of the lists computed by the GBA squares in every step without elimination

Algorithm 10 Pseudo code of improved GBA algorithm

INPUT: A $r \times n$ parity check matrix H, integer $t = 2^a$, integers $l_1, \ldots, l_a$ with $\sum_i l_i \leq r$, vector s, a prime power q.
OUTPUT: Index(es) index(x) such that the corresponding weighted sum equals s, or "No solution found".
SETUP: Add the first entry of all n columns of H to each list $L_1^1, \ldots, L_t^1$. Subtract s from all elements of L_t^1. Let $z = 0$.

for $step \leftarrow 1$ **to** a **do**
  $z \leftarrow z + l_{step}$
  for $j \leftarrow 1$ **to** 2^{a-step} **do**
    if $z = 0$ **then**
      $L_j^{step+1} = L_{2j-1}^{step} \oplus_1 L_{2j}^{step}$
    else
▷ If necessary, compute all bits from the list entries using equation (4.17); then continue with the regular GBA.
      $L_j^{step+1} = L_{2j-1}^{step} \oplus L_{2j}^{step}$
    end if
  end for
end for
if $|L_1^a| = 0$ **then**
  Return "No solution found"
end if
return index(L_1^a)

and remains constant during the elimination phase (except for the final list, the size of which is usually 1): Let $|L_i^1| = b$, and let j_0 be the first step in which elimination takes place, then the estimated list size in step j is

$$|L_i^j| = \begin{cases} 2^{2^{j-1}b} & j < j_0 \\ 2^u & \text{else} \end{cases},$$

where $2^{2^{j_0-2}b} \leq 2^u < 2^{2^{j_0-1}b}$. Since we have $a - j_0 + 1$ merge steps with elimination, we can estimate the total complexity of the unmodified GBA as

$$\sum_{i=1}^{j_0-1} 2^{2^{i-1}b} + (a - j_0 + 1)2^u.$$

The fraction of operations the modified GBA requires compared with the unmodified version is thus

$$F = \frac{\frac{\sum_{i=1}^{j_0-1} 2^{2^{i-1}b}}{r} + (a - j_0 + 1)2^u}{\sum_{i=1}^{j_0-1} 2^{2^{i-1}b} + (a - j_0 + 1)2^u}. \tag{4.18}$$

The best improvement, i.e. the smallest value of this fraction, is achieved for only one round of elimination, and for $2^u = 2^{2^{j_0-2}b}$:

$$F = \frac{\sum_{i=1}^{j_0-1} 2^{2^{i-1}b} + r2^{2^{j_0-2}b}}{r(\sum_{i=1}^{j_0-1} 2^{2^{i-1}b} + 2^{2^{j_0-2}b})}.$$

Since 2^u is very large, often in the order of 2^{30} to 2^{40}, we get a very good approximation

for F by neglecting all smaller list sizes:

$$F \approx \frac{(r+1)2^{2^{j_0-2}b}}{2r2^{2^{j_0-2}b}} = \frac{r+1}{2r}.$$

As r is in the order of 100 to more than 1000, $F \approx 1/2$. Therefore, in these cases the effect is an improvement factor of up to 2.

4.3.2. Examples

In this section, we will present several cryptosystems which can be attacked by GBA and where the attack benefits from our improvement. We start each section with a brief description of the cryptosystem and then compute the maximum improvement that can be gained using our technique for various parameters.

QD-CFS signature scheme

In 2009, Misoczki and Barreto [61] proposed QD binary Goppa codes to be used for the McEliece PKC, and in 2010 Barreto et al. presented the QD-CFS signature scheme [8]. Since every row of a dyadic matrix is generated from the first row by a permutation, we can apply the improved GBA. We illustrate this for the QD-CFS signature scheme. It is possible to attack a QD-McEliece cryptosystem as well using our improved GBA. However, since GBA would be very inefficient compared to other attacks (QD-McEliece has only a single solution), for instance ISD, we do not consider it here.

Let $h = (h_0, \ldots, h_{n-1})$ be the first row of a dyadic matrix. Then the i-th row can be computed by

$$\theta_i(h) = (h_{i \oplus j})_{j=0..n}.$$

From this we can compute

$$\Theta_i^{1-QD}(h) := (\theta_i \circ \theta_{i-1}^{-1})(h),$$

where the superscript 1-QD denotes that this is a QD matrix consisting of one dyadic block.
If H consists of several blocks of dyadic matrices, we can "glue" the corresponding permutations Θ together as we have done in the previous subsection. For example, in the 2-QD case, where the two blocks have h and g as their first rows, we have

$$\theta_i(h|g) = (h_{i \oplus j})_{j=0..n/2} | (g_{i \oplus j})_{j=0..n/2}$$

and Θ_i^{2-QD} similar to Θ_i^{1-QD} above.
Table 4.7 presents the maximum improvement factor for several parameter sets. All parameters refer to binary codes since codes over larger fields have a significantly smaller density (ratio of decodable words) and are therefore not suitable for CFS signatures. For these parameters, we achieve an improvement of up to 40%.

Note that the improvement factor is higher, the closer the maximum list sizes before and during elimination are (compare with equation (4.18) in the previous section).

Table 4.7.: Improvement of time and space complexity (written in $\log_2$) for QD-CFS.

n	r	t	Security level	Largest list size (in $\log_2$) Before elimination	During elimination	Improvement factor
30924	180	12	80	59.66	61	1.393
989724	240	12	100	79.66	81	1.394
31671168	300	12	120	99.66	101	1.395

Table 4.8.: Time and space complexity (written in $\log_2$) for the FSB hash function, assuming quasi-cyclic matrix and collision attack. FSB_{48} is a toy version.

	n	$2t$	r'	Security level	Largest list size (in $\log_2$) Before elimination	During elimination	Improvement factor
FSB_{48}	$3 \cdot 2^{17}$	$2 \cdot 24$	192	35.8	31.0	31	1.00
FSB_{160}	$5 \cdot 2^{18}$	$2 \cdot 80$	640	119.2	81.3	96	1.00
FSB_{224}	$7 \cdot 2^{18}$	$2 \cdot 112$	896	166.9	83.2	146	1.00
FSB_{256}	2^{21}	$2 \cdot 128$	1024	190.7	168.0	172	1.02
FSB_{384}	$23 \cdot 2^{16}$	$2 \cdot 184$	1472	281.0	164.2	229	1.00
FSB_{512}	$31 \cdot 2^{16}$	$2 \cdot 248$	1984	378.7	167.6	330	1.00

FSB hash function

The FSB authors suggest to use a *truncated* quasi-cyclic matrix in order to reduce the memory requirements and to increase the speed of the function. A truncated QC matrix is a block matrix consisting of blocks of size $r \times r$, where the upper left hand $r' \times r'$ submatrix of each block is cyclic. While the effect of our improvement is reduced because only part of the matrix is structured, it can still be applied to this type of structure.
Table 4.8 shows this effect, assuming a truncated quasi-cyclic matrix and a collision attack. Our numbers for the unmodified GBA will be a little lower that the original computation since we use the resulting list size as the complexity of the merge operation (as proposed in [60]) and don't include logarithmic factors.

For these parameters, the small improvement factor is due to two facts:

- The list size during elimination is much larger than before, so our technique applies to relative small lists and thus has a small impact.
- Elimination is applied to several merge steps instead of just one. Therefore, there are several high-cost steps onto which our technique can not be applied.

Single-solution applications

The applications above share the property that there are many possible solutions, and an attack needs to find any one of them. This fact makes the GBA very efficient since it allows to greatly decrease the size of the lists that need to be stored and computed, and

still expect to find at least one solution.
Other applications only have a single solution. Examples for these applications include:

- **QD-McEliece:** In [61], QD codes were proposed to be used for the McEliece encryption scheme. In contrast to the QD-CFS scheme mentioned above, the parameters were chosen such that there exists only one solution for each instance.
- **QC-McEliece:** In [10], the authors propose to use QC alternant codes for the McEliece PKC in order to reduce the size of the public key. A QC code can be defined by a QC matrix, i.e. a block matrix where each block is cyclic.
- **QC Stern-ID:** Gaborit and Girault showed in [38] how to use random QC codes for the identification scheme Stern presented in 1993 [89].
- **SYND:** The SYND stream cipher [39] was proposed in 2007 by Gaborit, Lauradoux and Sendrier. It is based on the SD problem and uses QC codes.

Against these, GBA is much less efficient since any reduction in the list size can eliminate the solution. Depending on the elimination strategy, GBA will perform between the two extremes:

- No elimination, i.e. testing *all* combinations and finding the solution with certainty (meaning large list sizes)
- Maximum elimination, which essentially means randomly trying combinations (small list size, but large number of iterations)

Depending on the parameters, the work factor for GBA can be larger than 2^{300} operations, compared to 2^{85} for other attacks (in this case, ISD against the QC Stern-ID). Our improvement could be applied in these cases as well, but due to the reasons above, it does not make sense to use GBA.

4.4. Statistical decoding of codes over $\mathbb{F}_q$

Statistical decoding was introduced in 2001 by Al Jabri [45] and improved by Overbeck in 2006 [72]. While Al Jabri claimed that statistical decoding can be used effectively against the McEliece cryptosystem, Overbeck showed that far more precomputation is required than expected by Al Jabri, and that therefore the time as well as the memory requirements are much higher compared with other attacks. However, statistical decoding is quite efficient in decoding short codes (i.e. codes with a small length n) and can even be faster than attacks based on ISD or the GBA (see the recent resources [16, 67] and [60] for more information on these attacks).

The basic principle of statistical decoding is as follows: After receiving a codeword with error $c = mG + e$, where m and e are unknown, a precomputed set $H_w \subseteq \mathcal{C}^\perp$ is used as a mask to obtain information about e. Since $GH_w^{\mathrm{T}} = 0$, we have

$$H_w c^{\mathrm{T}} = H_w e^{\mathrm{T}}.$$

Al Jabri showed that if $hc^{\mathrm{T}} = 1$ for some $h \in H_w$ then the non-zero bits of h give some information about the non-zero bits of e. Overbeck has improved this algorithm by also using the vectors h where $hc^{\mathrm{T}} = 0$ to gain information about e.
We will briefly describe Overbeck's algorithm and then generalize it to codes over non-binary fields $\mathbb{F}_q$. Our algorithm for statistical decoding over $\mathbb{F}_q$ is shown in Algorithm 13 (page 89). In Section 4.4.3 we describe two techniques that allow a further improvement of our algorithm by making use of additional structure (quasi-cyclic matrices and regular words). In the subsequent sections we conclude by providing experimental results and compare our statistical decoding algorithm with ISD.

4.4.1. Binary statistical decoding

Let $\mathcal{C}$ be an (n, k, t) code over $\mathbb{F}_q$, $w < n/2$ be an integer, and $H_w \subseteq \mathcal{C}^{\perp}$ a sufficiently large subset of the dual space of $\mathcal{C}$, where $\forall h \in H_w : \mathrm{wt}(h) = w$. Given a word $c = x + e$, where $x = mG \in \mathcal{C}$ and $\mathrm{wt}(e)$ is small, the algorithm attempts to find e.

For every $h \in H_w$, we have an *odd error detection* at bit i if $hc^{\mathrm{T}} = 1$ and $h_i = 1$, and an *even error detection* at bit i if $hc^{\mathrm{T}} = 0$ and $h_i = 1$. In each case we can compute the probabilities that e contains an error at bit i. In the case of an odd error detection, the probabilities p_w^+ and q_w^+ that $e_i = 1$ and $e_i = 0$, respectively, are

$$p_w^+ = \frac{\sum_{j \text{ odd}}^{\leq t} \binom{n-t}{w-j}\binom{t-1}{j-1}}{\sum_{j \text{ odd}}^{\leq t} \binom{n-t}{w-j}\binom{t}{j}}, \quad q_w^+ = \frac{\sum_{j \text{ odd}}^{\leq t} \binom{n-t-1}{w-j-1}\binom{t}{j}}{\sum_{j \text{ odd}}^{\leq t} \binom{n-t}{w-j}\binom{t}{j}}.$$

Let $v_{y,w}^+ = |\{h \in H_w : hc^{\mathrm{T}} \neq 0\}|$. For every bit i, the random variable

$$\frac{1}{v_{y,w}^+} \sum_{h \in H_w} (hc^{\mathrm{T}} \mod 2) h_i$$

is the relative frequency estimate for p_w^+ or q_w^+, depending on whether i is an error position of e. The variance of this random variable is $(\sigma_w^+)^2 = p_w^+(1 - p_w^+)/v_{y,w}^+$. Thus, for H_w large enough, Algorithm 11 allows to recover m.

Algorithm 11 Al Jabri's algorithm for binary statistical decoding.

INPUT: Generator matrix G for an (n, k, t) code, $H_w \subseteq \mathcal{C}^{\perp}$ and $c \in \{0, 1\}^n$
OUTPUT: $m \in \{0, 1\}^k$ such that $\mathrm{wt}(c - mG) \leq t$

$v \leftarrow \sum_{h \in H_w} (hc^{\mathrm{T}} \mod 2) h \in \mathbb{Z}^n$

Choose $I = \{\text{positions of the } k \text{ smallest entries of } v\}$ s.t. $G_{.I}$ is invertible

Return $m \leftarrow c_I G_{.I}^{-1}$

Overbeck improved this algorithm in two ways. First, even error detections are used as well, allowing to extract significantly more information from a given set H_w. Second, the algorithm is no longer restricted to a fixed value of w. Instead, it allows a range for w,

and the information extracted from the different sets H_w is combined in the end.
In case of an even error detection, the corresponding probabilities p_w^- and q_w^- are given by

$$p_w^- = \frac{\sum_{2\leq j \text{ even}}^{\leq t} \binom{n-t}{w-j}\binom{t-1}{j-1}}{\sum_{j \text{ even}}^{\leq t} \binom{n-t}{w-j}\binom{t}{j}}, \quad q_w^- = \frac{\sum_{j \text{ even}}^{\leq t} \binom{n-t-1}{w-j-1}\binom{t}{j}}{\sum_{j \text{ even}}^{\leq t} \binom{n-t}{w-j}\binom{t}{j}}.$$

Consequently, $v_{y,w}^- = |\{h \in H_w : hc^{\mathrm{T}} = 0\}|$, and the relative frequency estimates are given by

$$\frac{1}{v_{y,w}^-} \sum_{h \in H_w} (1 - hc^{\mathrm{T}} \mod 2) h_i.$$

Algorithm 12 summarizes the improved algorithm. Note that v is defined as $v = \sum_{w-b}^{B} a_w v_w + \sum_{w=b}^{B} a_{w+B} v_{w+B}$, where each $a_i \in \{0,1\}$, i.e. not all partial results need to be combined in the end.

Algorithm 12 Overbecks's improved algorithm for binary statistical decoding.

INPUT: Generator matrix G for an (n,k,t) code $\mathcal{C}$, $H = \bigcup_{w=b}^{B} H_w \subseteq \mathcal{C}^{\perp}$ and $c \in \{0,1\}^n$
OUTPUT: $m \in \{0,1\}^k$ such that $\mathrm{wt}(c - mG) \leq t$

Let $\mathbf{1} = (1, \ldots, 1) \in \{0,1\}^n$
for $w = b \to B$ **do**
  $(\sigma_w^+)^2 = p_w^+(1-p_w^+)v_{y,w}^+$
  $(\sigma_w^-)^2 = p_w^-(1-p_w^-)v_{y,w}^-$

  $v_w \leftarrow \sum_{h \in H_w} (hc^{\mathrm{T}} \mod 2)(h - p_w^+\mathbf{1})/\sigma_w^+ \in \mathbb{R}^n$
  $v_{w+B} \leftarrow -\sum_{h \in H_w} (1 - hc^{\mathrm{T}} \mod 2)(h - p_w^-\mathbf{1})/\sigma_w^- \in \mathbb{R}^n$
end for

for all binary combinations v of the different v_i **do**
  Choose $I = \{$positions of the k smallest entries of $v\}$ s.t. $G_{\cdot I}$ is invertible
  $m \leftarrow c_I G_{\cdot I}^{-1}$
  if $\mathrm{wt}(c - mG) \leq t$ **then**
    Return m
  end if
end for

4.4.2. Statistical decoding over $\mathbb{F}_q$ (for $q > 2$)

The first thing to note when considering codes over non-binary fields $\mathbb{F}_q$ is that the error positions of the secret vector e now take values in $\mathbb{F}_q \backslash \{0\}$. However, we are only interested to find k error-free positions such that the corresponding generator matrix $G_{\cdot I}$ is invertible, so we do not have to find those error values.
Secondly, we note that the definition of odd error detection needs to be changed, since $hc^{\mathrm{T}} \in \mathbb{F}_q \backslash \{0\}$ as well: We define odd error detection at bit i as the case when $hc^{\mathrm{T}} \neq 0$ and $h_i \neq 0$. The reason is that since

$$\forall x \in \mathbb{F}_q \backslash \{0\} : h(xc)^{\mathrm{T}} = x(hc^{\mathrm{T}}) \quad \text{and} \quad hc^{\mathrm{T}} \neq 0 \Leftrightarrow x(hc^{\mathrm{T}}) \neq 0, \tag{4.19}$$

all values of hc^{T} have the same probability, and they are independent of the value of h_i. Finally, the original algorithm adds up the vectors h in order to compute the relative frequencies of p_w^+ and q_w^+. Doing the same over $\mathbb{F}_q$ would disturb these frequencies because entries of h greater than 1 bias the computation. Instead, we add $\Theta(h) = (\theta(h_1), \ldots, \theta(h_{n-k}))$, where

$$\theta : \mathbb{F}_q \to \mathbb{F}_2, x \mapsto \begin{cases} 0 & x = 0 \\ 1 & \text{else} \end{cases}.$$

In order to proceed, we introduce some notation. Consider the case where e and a vector h are simultaneously non-zero in exactly i bits. In contrast to the binary case, we don't have the equivalence i even $\Leftrightarrow hc^{\mathrm{T}} = 0$. Since every non-zero value of hc^{T} has the same probability (and occurs the same number of times when $H_w = \mathcal{C}^{\perp}$), the quantities

$$\mathfrak{C}(q,i) = \left\lceil \frac{(q-1)^i}{q} \right\rfloor \tag{4.20}$$

$$\mathfrak{C}'(q,i) = (q-1)^i - \left\lceil \frac{(q-1)^i}{q} \right\rfloor \cdot (q-1) \tag{4.21}$$

reflect the relative frequencies of non-zero and zero values of hc^{T}, respectively, where $\lceil \rfloor$ denotes rounding to the nearest integer. They are well-defined since $(q-1)^i/q \in \mathbb{N} + \{0.5\}$ can only occur for $q = 2$.

Using equation (4.19), we can calculate the respective probabilities.

$$p_w^{++} = \frac{\sum_{j=1}^{t} \mathfrak{C}(q,j)\binom{t-1}{j-1}\binom{n-t}{w-j}(q-1)^{w-j}}{\sum_{j=1}^{t} \mathfrak{C}(q,j)\binom{t}{j}\binom{n-t}{w-j}(q-1)^{w-j}} \tag{4.22}$$

$$q_w^{++} = \frac{\sum_{j=1}^{t} \mathfrak{C}(q,j)\binom{t-1}{j}\binom{n-t}{w-j}(q-1)^{w-j}}{\sum_{j=1}^{t} \mathfrak{C}(q,j)\binom{t}{j}\binom{n-t}{w-j}(q-1)^{w-j}} \tag{4.23}$$

$$p_w^{--} = \frac{\sum_{j=0}^{t} \mathfrak{C}'(q,j)\binom{t-1}{j-1}\binom{n-t}{w-j}(q-1)^{w-j}}{\sum_{j=0}^{t} \mathfrak{C}(q,j)\binom{t}{j}\binom{n-t}{w-j}(q-1)^{w-j}} \tag{4.24}$$

$$q_w^{--} = \frac{\sum_{j=0}^{t} \mathfrak{C}'(q,j)\binom{t-1}{j}\binom{n-t}{w-j}(q-1)^{w-j}}{\sum_{j=0}^{t} \mathfrak{C}(q,j)\binom{t}{j}\binom{n-t}{w-j}(q-1)^{w-j}} \tag{4.25}$$

Consequently, we redefine

$$v_{y,w}^{++} = |\{h \in H_w : hc^{\mathrm{T}} \neq 0\}| \tag{4.26}$$

$$v_{y,w}^{--} = |\{h \in H_w : hc^{\mathrm{T}} = 0\}| \tag{4.27}$$

Since we sum up $\Theta(h)$ (instead of h), the variance of v remains unchanged, i.e.

$$\sigma_w^{++} = p_w^{++}(1 - p_w^{++})v_{y,w}^{++}.$$

Algorithm 13 is the generalized version for statistical decoding over $\mathbb{F}_q$.

Algorithm 13 Generalized algorithm for statistical decoding over a non-binary field $\mathbb{F}_q$.

INPUT: Generator matrix G for an (n, k, t) code $\mathcal{C}$, $H = \bigcup_{w=b}^{B} H_w \subseteq \mathcal{C}^\perp$ and $c \in \mathbb{F}_q^n$
OUTPUT: $m \in \mathbb{F}_q^k$ such that $\mathrm{wt}(c - mG) \leq t$

Let $\mathbf{1} = (1, \ldots, 1) \in \{0, 1\}^n$
for $w = b \to B$ **do**
  $(\sigma_w^{++})^2 = p_w^{++}(1 - p_w^{++})v_{y,w}^+$
  $(\sigma_w^{--})^2 = p_w^{--}(1 - p_w^{--})v_{y,w}^-$

  Let $H_w^+ = \{h \in H_w : hc^{\mathrm{T}} \neq 0\}$ and $H_w^- = H_w \backslash H_w^+$
  $v_w^+ \leftarrow \sum_{h \in H_w^+} (\Theta(h) - p_w^{++}\mathbf{1})/\sigma_w^{++} \in \mathbb{R}^n$
  $v_{w+B}^+ \leftarrow -\sum_{h \in H_w^-} (\Theta(h) - p_w^{--}\mathbf{1})/\sigma_w^{--} \in \mathbb{R}^n$
end for

for all binary combinations v^+ of the different v_i^+ **do**
  Choose $I = \{$positions of the k smallest entries of $v\}$ s.t. $G_{\cdot I}$ is invertible
  $m \leftarrow c_I G_{\cdot I}^{-1}$
  if $\mathrm{wt}(c - mG) \leq t$ **then**
    Return m
  end if
end for

4.4.3. Exploiting additional structure

Many types of additional structure have been proposed in code-based cryptography in order to reduce the public key size or to increase efficiency. Algorithm 13 allows to exploit various types of such structures. We will give two examples and briefly describe the corresponding techniques:

(Quasi-)cyclic matrices

We will describe the technique using cyclic matrices, but it applies to QC matrices as well, and also to other types of structured matrices like QD matrices.

Let $\gamma(v)$ denote the cyclic shift of vector v. A cyclic code $\mathcal{C}$ allows to choose a cyclic parity check matrix H. Therefore, for every $h \in \mathcal{C}^\perp$, $\gamma(h) \in \mathcal{C}^\perp$. This means that we can restrict the precomputed matrix H_w to vectors that are not cyclic shifts of one another, and in the course of running Algorithm 13, test $h \in H_w$ as well as all cyclic shifts of h against the vector c. As a result, while the run time of the actual algorithm is unchanged, the size of the precomputed set H_w can be decreased by a factor of up to n.

Regular words

If it is known that the solution is a regular word, the decoding algorithm can be modified as follows. When choosing the set I, we add the additional condition that I must not contain those indices corresponding to a whole block; in other words, for all i with $1 \leq i \leq t$,

$$\left\{ \frac{(i-1)n}{t} + 1, \ldots, \frac{in}{t} \right\} \subsetneq I.$$

If the values in v are such that a this would happen, the largest value of v in this block is ignored and the index of the next smallest value of v is added to I instead. This modification slightly increases the chance of decoding successfully, or to achieve the same success probability with a slightly smaller size of H_w.

4.4.4. Experimental results

In this section, we derive a relation between the success probability of our algorithm and the size of the set H_w. We restrict our analysis to the case $b = B$ and the use of odd error detection only, since we were not able to prove the results for the general case. Our results therefore establish lower bounds for the success probability of our algorithm. We conclude our analysis with experimental results of statistical decoding over $\mathbb{F}_q$.

The success probability of Algorithm 13 depends on w and the size of the set H_w, and it increases with $|H_w|$. However, the size of H_w is bounded from above:

Lemma 4.4.1 (Upper bound for the size of H_w)**.** *Let $\mathcal{C}$ be an $[n, k]$ code over $\mathbb{F}_q$. An upper bound for the size of H_w is*

$$|H_w| \leq \binom{n}{w}(q-1)^w q^{-k}.$$

Proof. The set H_w is a subset of the dual code $\mathcal{C}^\perp$, which is a $[n, n-k]$ linear code. In [27], the authors generalize the weight distribution results from [53] to random codes over $\mathbb{F}_q$: In the $[n, n-k]$ code $\mathcal{C}^\perp$, the ratio of codewords of weight w to words of weight w approaches the constant $Q = q^{-k}$ as w becomes large. Since there are $\binom{n}{w}(q-1)^w$ words of weight w, the lemma follows. □

We will derive the success probability of our algorithm under two assumptions, and then provide arguments for the correctness of these assumptions:

1. $v_{y,w}^{++} = \frac{q-1}{q}|H_w|$
2. Half the values of v_j, for j the non-error positions, are below their mean of $p_w^{++}v_{y,w}^{++}$, where v_j denotes the j-th bit of v_w^+.

Proposition 4.4.2. *Under the above assumptions, Algorithm 13 for statistical decoding in an (n, k, t) code over a non-binary field $\mathbb{F}_q$ has a success probability of 0.95^t when a precomputed set of size*

$$|H_w| = 2.72\frac{q}{q-1} \cdot \frac{p_w^{++}(1-p_w^{++})}{(p_w^{++} - q_w^{++})^2}$$

is used.

In order to compute the success probability $\mathcal{P}$, there needs to be a value δ such that the following conditions hold (these conditions were introduced in [72]):

1. For every error position i:
$$v_i > (p_w^{++} - \delta)v_{y,w}^{++}.$$

2. There are at least k non-error positions j such that:

$$v_j < (p_w^{++} - \delta)v_{y,w}^{++}.$$

Using assumption (1) from above, the probability $\mathcal{P}$ that a certain δ satisfies the first condition is

$$\mathcal{P} = \Phi(\delta/\sigma_w^{++})^t = \Phi\left(\delta\sqrt{\frac{(q-1)|H_w|}{qp_w^{++}(1-p_w^{++})}}\right)^t,$$

where Φ refers to the standard normal distribution. Therefore, we get the following condition on $|H_w|$:

$$(\Phi^{-1}(\mathcal{P}^{1/t}))^2\delta^{-2}\frac{q}{q-1}p_w^{++}(1-p_w^{++}) \leq |H_w| \leq \binom{n}{w}(q-1)^w q^{-k}. \tag{4.28}$$

Following from assumption (2) is that a δ satisfying both conditions above is expected to be smaller than $|p_w^{++} - q_w^{++}|$. Thus, we expect a success probability of 0.95^t when a set of size

$$|H_w| = 2.72\frac{q}{q-1} \cdot \frac{p_w^{++}(1-p_w^{++})}{(p_w^{++} - q_w^{++})^2}$$

is used (since $\Phi^{-1}(0.95)^2 \approx 2.72$). Note that this size differs by a factor of $\frac{q}{2(q-1)}$ from the binary case.

Validation of the assumptions

Since non-zero multiples of vectors in $\mathcal{C}^\perp$ are also in $\mathcal{C}^\perp$ and $\forall a \in \mathbb{F}_q, a \neq 0 : hc^T \neq 0 \Rightarrow ahc^T \neq 0$, the expected frequency of every non-zero value in the computations hc^T, $h \in H_w$, is identical. The expected frequency of zeros in these computations is nearly the same (see equations (4.20) and (4.21)). Therefore, the number of non-zero results is expected to be $|H_w|(q-1)/q$.
For parameters used in cryptography, p_w^{++} is close to 0.5 (otherwise a small set H_w would be sufficient and decoding was easy). Also, due to the large size of H_w, $v_{y,w}^{++}$ is large. Therefore, we expect the values of v_j to be close to their mean value and to be above and below it approximately the same number of times (due to the law of large numbers).

In Table 4.9 we present experimental results obtained using our implementation in Maple.

The results show that in many cases our algorithm decodes successfully, even though the number of sample vectors $|H|$ was not very large. Also, the success probability can be increased by using a larger weight spectrum $B - b$. However, this increases the complexity of testing all binary combinations v^+. Note that the success probability seems to be independent of the field size q; this is to be expected, since we are only searching for the (non-)error *positions*, not their values. This is an advantage compared with other algorithms like information set decoding, where the algorithm complexity grows significantly with q (more than the impact of q-ary arithmetic, which applies in our case as well).

Table 4.9.: Experimental results of using Algorithm 13 to decode t errors in an (n, k) code over $\mathbb{F}_q$. We ran several thousand decoding attempts, each using a sample of size $|H| = |\bigcup_{w=b}^{B} H_w| = 100$.

(n, k, t)	q	b	B	Successful decodings
$(64, 40, 4)$	3	44	46	30.6%
	5	51	53	29.8%
	7	56	58	36.1%
	11	58	60	35.8%
	13	59	61	42.7%
	53	61	63	29.4%
$(128, 72, 8)$	3	84	88	18.1%
	5	100	104	22.9%
	7	108	112	23.0%
	11	115	119	32.1%
	13	117	121	27.8%
	53	123	127	37.8%

Also, note that larger field sizes require larger values w when computing the sets H_w. This is due to the fact that the weight distribution of codes over different fields is not identical. For the above fields $\mathbb{F}_q$ with $q \in \{3, 5, 7, 11, 53\}$, the weight distributions are shown in Figure 4.3. Those distributions are derived from Cheung's result that in an (n, k) code over $\mathbb{F}_q$, the ratio of codewords of weight u to words of weight u is very close to

$$q^{-(n-k)}. \tag{4.29}$$

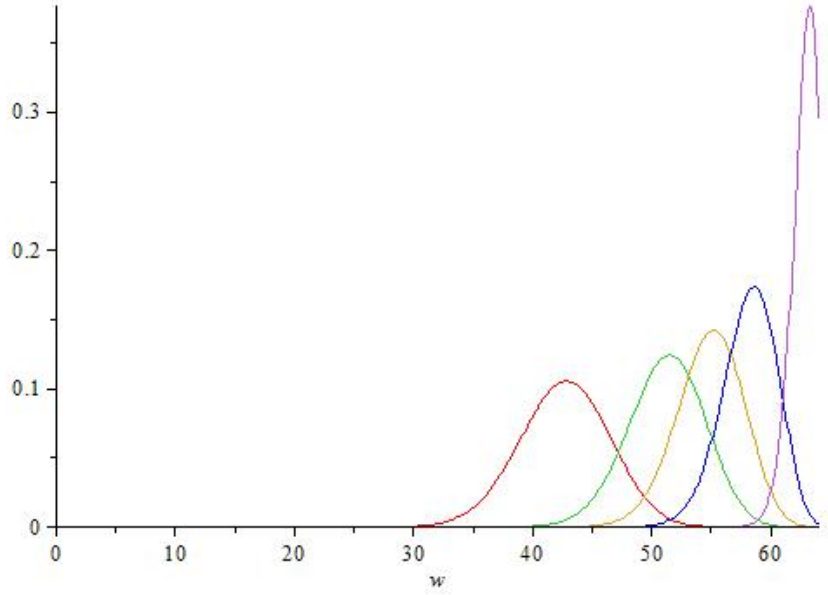
The optimal choice of b and B is difficult to compute: since vectors h of smaller weight can provide more information about the error positions of e, a smaller set H_w is sufficient to achieve a given success probability, but it is more difficult to precompute this set if there exist fewer vectors of this weight in the code. A good value (or range of values) can be estimated using Equations (4.28) and (4.29).

4.4.5. Comparison with ISD

Information set decoding (ISD) is based on a decoding algorithm by Prange [79]. Improved versions of this attack, e.g. [74] achieve complexities close to theoretical lower bounds [67, 68].

For those parameters typically used today in code-based cryptography, ISD is much faster than statistical decoding. However, the complexity of ISD increases significantly with the field size q. To estimate the value of q for which statistical decoding becomes faster than ISD, we will compare our algorithm with the one in [74].
In the case of statistical decoding, the largest part of the complexity is due to the generation of the sample sets H_w, so we will restrict our analysis to this. Our algorithm is not fully optimized; for example, the sets H_w are sampled essentially randomly, instead of using

Figure 4.3.: Weight distribution of $[64, 40]$ codes over $\mathbb{F}_q$ for $q \in \{3, 5, 7, 11, 53\}$. Larger fields correspond to higher weights.



a generalized version of ISD to sample the vectors. We will therefore estimate the total work factor of statistical decoding by

$$\mathrm{WF}_{\mathrm{SD}} \approx \frac{2n(n-k)|H|}{F \cdot P},$$

where $H = \cup_w H_w$, F is the fraction of codewords c with $b \leq \mathrm{wt}(c) \leq B$, and P is the success probability of decoding. The factor of $2n(n-k)$ reflects the fact that our sampling algorithm requires $n(n-k)$ multiplications and additions.
Note that both algorithms estimate the number of q-ary operations (instead of binary operations), so the results are comparable.

For the $(64, 40)$ code over $\mathbb{F}_3$, ISD requires $2^{13.9}$ operations, compared with $2^{20.2}$ for our algorithm. Increasing q, we find that ISD is slower than statistical decoding for $q \geq 1201$. In the case of the $(128, 72)$ code and $q = 3$, the number of operations is $2^{18.3}$ for ISD and $2^{22.0}$ for statistical decoding. Here, $q \geq 233$ is sufficient to make our algorithm the more efficient one.

4.5. "Cross-area" attacks

In this section, we describe and analyze three "cross-area" decoding attacks, i.e. lattice-algorithms against code-based cryptosystems and code-based algorithms against lattice cryptosystems:

- Sieving algorithm against code-based cryptosystems
- Enumeration algorithm against code-based cryptosystems
- ISD against lattice-based cryptosystems

We turns out that the sieving and ISD attacks do work, i.e. they can be used to solve the hard problem required to run a decoding attack. However, in both cases the "cross-area" attack is slower than the best known decoding attack from the within the other area. We are not able to modify the enumeration attack to work against code-based schemes. In the following, we will describe each approach and our findings in detail.

4.5.1. Sieving algorithm against code-based cryptosystems

In [59], Micciancio and Voulgaris presented a new sieving-based attack against lattice-based cryptosystems. Since our attack is a modification of their *Gauss sieve* attack (*List sieve* has a proven upper complexity bound, but is much slower in practice), we will briefly describe the idea of this type of attack. The original attack solves the following problem:

Problem 4.5.1 (Shortest Vector Problem (SVP))**.** *Given an n-dimensional lattice $\mathcal{L}$, i.e. a linear subspace of $\mathbb{Z}^n$, and a metric $||\cdot||$; find a vector $x \in \mathcal{L}$ of minimal length*

$$||x|| = \min_{y \in \mathcal{L} \setminus \{0\}} ||y||.$$

The Sieving algorithm starts with an empty list L. In each iteration, a random vector $v_{\text{new}} \in \mathcal{L}$ is sampled. This vector is used to try to reduce the length of the vectors in the list L by substituting $v \in L$ with $v - v_{\text{new}}$ if the latter is smaller. In the same way, v_{new} is reduced against the vectors in the list. If the resulting vector v'_{new} is non-zero, it is added to the list L. These steps are repeated until a vector of sufficiently small length is found.

In order to allow this algorithm to find a small codeword, we have to modify two parts: Firstly, the computations are performed over a finite field $\mathbb{F}_q$ instead of over $\mathbb{Z}$. Secondly, the Hamming weight has to be used instead of the (usually Euclidean) metric in the lattice case. Algorithms 14 and 15 summarize the steps of the modified algorithm.

The original lattice algorithm has a time complexity of $2^{0.52n}$, and space complexity of $2^{0.2n}$. Without being able to prove this, we expect a similar behaviour of the modified version.

Algorithm 14 GaussSieve($G, q, \langle t_1, t_2 \rangle$)

$L \leftarrow \{0\}$, $S \leftarrow \{\}$, $K \leftarrow 0$
while $K < c$ **do**
  if S is not empty **then**
    $v_{\text{new}} \leftarrow$ S.pop()
  else
    $v_{\text{new}} \leftarrow$ Sample(G, q, $\langle t_1, t_2 \rangle$)
  end if
  $v_{\text{new}} \leftarrow$ GaussReduce(v_{new}, L, S, q)
  if $v_{\text{new}} = 0$ **then**
    $K \leftarrow K + 1$
  else
    $L \leftarrow L \cup \{v_{\text{new}}\}$
  end if
end while

Algorithm 15 GaussReduce(p, L, S, q)

while $\exists v \in L : (\text{wt}v \leq \text{wt}p) \wedge (\text{wt}(p - v) \mod q \leq \text{wt}p)$ **do**
  $p \leftarrow (p - v) \mod q$ ▷ Reduce p
end while
while $\exists v_i \in L : \text{wt}v_i > \text{wt}p \wedge \text{wt}(v_i - p) \mod q \leq \text{wt}v_i$ **do**
  $L \leftarrow L \setminus \{v_i\}$ ▷ Remove v_i from list L
  S.push($(v_i - p) \mod q$) ▷ Push reduced v_i on the stack
end while
return p

Experimental results

In the following, we test our implementation using different parameter sets, and compare the results with the corresponding complexity of running an ISD-based attack. To keep the comparison fair and as independent from implementation details as possible, we count in both cases only the number of operations for the most important steps. For ISD, the details are described in Section 4.1. For sieving, we count the following operations:

- Sampling a new vector
- Comparing $\text{wt}(p - v \mod q) \leq \text{wt}(p)$ and comparing $\text{wt}(v - p \mod q) \leq \text{wt}(v)$

This means, do not count

- Push on or pop from stack
- Other memory operations like fetching an entry from the list
- Adding an entry to or removing from the list

Table 4.10.: Comparison of our sieving implementation with ISD using binary parameters. The number of operations is in $\log_2$.

		Operations		
n	t	sieving	ISD	Ratio sieving/ISD
56	8	22.6	16.0	1.41
70	9	25.3	17.4	1.46
80	9	27.1	17.7	1.53
100	11	30.4	20.1	1.51
120	13	35.8	22.4	1.60

The results are shown in Table 4.10.

To test the effect of larger fields, we also use different values of q. However, for growing q, sieving seems to become worse compared with ISD. This behaviour is no surprise: Even though we have no formal argument for this, we believe that the runtime of the sieving algorithm generalizes from $2^{0.34n}$ in the binary case to $q^{0.34n}$ in the q-ary case. Thus, the increase is $(q/2)^{0.34n}$.
For ISD, however, the birthday step gets slower by a factor of only $(q/2)^p < (q/2)^{0.34n}$, and the probability to find the right permutation remains constant.

Asymptotic complexity of sieving compared with ISD

In cryptography, we are interested in larger parameters than those that can be tested experimentally. It is therefore important to consider the asymptotic complexity of both algorithms when we increase the size of the parameters.

As mentioned above, we assume a time complexity of $2^{0.52n}$ and a space complexity of $2^{0.2n}$ for the modified sieving algorithm. As mentioned earlier, ISD-like algorithms have an approximate complexity of

$$C(n, R) = a(n)2^{-t(n,R)\log_2(1-R)},$$

where $a(n)$ is some polynomial in n and $R = k/n$ the code rate. For Goppa codes with a code rate of $R = 1 - \exp(-1) = 0.63$, we have a complexity of

$$C(n, R) = a(n)2^{0.53n/\lceil \log n \rceil},$$

since in this case $k = n - t * \lceil \log_2(n) \rceil$. These complexities are supported by Table 4.10 (page 96): Choosing $a(n)$ such that the ratio of sieving over ISD is correct for $n = 56$, we find for the other ratios 1.49, 1.53, 1.61, and 1.68.

This shows that the asymptotic complexity of the modified sieving algorithm is higher than the complexity of ISD. In addition to that, the space complexity of sieving is exponential, whereas that of ISD is quadratic.

4.5.2. Enumeration algorithm against code-based cryptosystems

This section covers the second attack, enumeration using extreme pruning [41]. In contrast to the sieving-based attack above, we are not able to modify this algorithm to an attack against code-based cryptosystems. We will briefly describe the idea of the algorithm, and then show why the modified version fails.

Lattice enumeration using (extreme) pruning

Enumeration algorithms attempt to solve the SVP, i.e. find the shortest vector in a given lattice $\mathcal{L} \in \mathbb{Z}^n$. Let $\mathcal{B} = (b_1, \ldots, b_n)$ be a basis of $\mathcal{L}$, then every lattice vector v can be represented as a linear combination of basis vectors: $v = \sum_i x_i b_i$, where $x_i \in \mathbb{Z}$. The most basic form of an enumeration algorithm performs an exhaustive search through all those linear combinations of basis vector and thus finds the shortest vector in exponential time. This is possible because the (originally infinite) search space $\mathbb{Z}^n$ can be restricted by using upper bounds on the norm of the shortest vector. See [41] for details.

The idea of (extreme) pruning is to make this algorithm probabilistic and only enumerate those branches of the search tree that have the greatest probability of containing the solution. This leads to an exponential speed-up of the total expected runtime. Let $v \in \mathcal{L}$ be the shortest vector, then v can be written as a linear combination of basis vectors $v = \sum_i v_i b_i$, where the v_i are the unknown coefficients. Let $\mathcal{B}^* = (b_1^*, \ldots, b_n^*)$ be a Gram-Schmidt reduced basis of $\mathcal{B}$, then v can also be written as

$$v = \sum_{i=1}^{n} v_i b_i = \sum_{i=1}^{n} v_i \left(b_i^* + \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \right) = \sum_{j=1}^{n} \left(v_j + \sum_{i=j+1}^{n} \mu_{i,j} v_i \right) b_j^*, \tag{4.30}$$

where the $\mu_{i,j} \in \mathbb{Q}$ form the lower-triangular matrix computed by the Gram-Schmidt orthogonalization. Let π_i be the orthogonal projection onto $(b_1, \ldots, b_{i-1})^{\perp}$. This allows to compute the norms of the projections of v as

$$||\pi_{n+1-k}(v)||^2 = \sum_{j=n+1-k}^{n} \left(v_j + \sum_{i=j+1}^{n} \mu_{i,j} v_i \right)^2 ||b_j^*||^2, 1 \leq k \leq n. \tag{4.31}$$

Since the norm of the projection $\pi_i(v)$ only depends on those unknown vectors v_j with $j \geq i$, the algorithm can apply bounds on the v_i *one at a time*. If R is the upper bound on v (which is either known or can be estimated, e.g. using the Gaussian heuristic), then the algorithm first applies the bound

$$0 \leq v_n \leq \frac{R}{||b_n^*||}.$$

The possible values of v_n form the first level of the search tree. For every value of v_n, the second level is formed by using equation (4.31) to apply a bound on v_{n-1} and so on. The resulting tree is the search tree for the (extreme) pruning algorithm.

Using (extreme) pruning against code-based cryptosystems

Equation (4.31) is crucial for the (extreme) pruning enumeration algorithm since it allows to estimate the norm of a vector $v = \sum_i v_i b_i$ using only a small number of v_i. As in the previous section, if we want to apply the algorithm on a code $\mathcal{C}$ generated by G, we need to estimate the Hamming weight of v instead of the norm. In order for equation (4.31) to work when we substitute the norm with the Hamming weight, we require an analogue to the Gram-Schmidt basis b_i^*, namely a generator matrix G^* for $\mathcal{C}$ such that $\mathrm{wt}(x+y) = \mathrm{wt}(x)+\mathrm{wt}(y)$ (in the lattice case, this works because the b_i^* are orthogonal). Unfortunately, in general it is not possible to construct such a basis, as the following binary example shows

$$G = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}.$$

We have implemented an algorithm to test the behaviour, but the estimates of the weight of v seem completely random, so pruning can not be used to improve the efficiency of enumeration.

4.5.3. ISD against lattice-based cryptosystems

In this section we analyze the third attack, ISD against lattice-based cryptosystems. In order to use ISD in this context, three issues have to be addressed: Defining the search space, modifying the target function from small Hamming weight to small Euclidean length, and estimating the number of non-zero entries which the ISD algorithm attempts to find. We will describe our approach to solve these issues and the associated issues. For the remainder of this section, let $\mathcal{L}$ denote the lattice for which we want to solve the SVP. Let the norm $||\cdot||$ be a norm commonly used in lattice-based cryptography, i.e. the 1-norm, Euclidean norm, or maximum-norm.

Defining the search space

In the collision-search step of (the unmodified) ISD, for every non-zero position the algorithm has to tests all values in $\mathbb{F}_q$. Lattices, however, are a subset of $\mathbb{Z}^n$ and therefore do not have a finite number of such values to test. If a bound on the shortest lattice vector is known, we can use the following proposition:

Proposition 4.5.2. *If the p-norm of the smallest lattice vector $v = (v_1, \ldots, v_n)$ is upper bounded by an integer b, then every entry v_i of v is upper bounded by b.*

Proof. Using the triangle inequality, we have $\forall i : b \geq ||v||_p = \left(\sum_{j=1}^{n} |v_j|^p\right)^{1/p} \geq |v_i|$. □

However, this bound is not very tight. In some cases, we can do much better.

Definition 4.5.3 (Ajtai lattice)**.** Let $u = (u_1, \ldots, u_n)$ be a sequence of vectors, where $u_i \in \mathbb{Z}_q^n$ and $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$. The Ajtai-lattice $\Lambda(u, q)$ is defined as

$$\Lambda(u, q) = \{x \in \mathbb{Z}^n : u_i x^{\mathrm{T}} = 0 \mod q \text{ for all } i\}.$$

Proposition 4.5.4. *If $\mathcal{L}$ is an Ajtai-lattice $\Lambda(u, q)$, then every entry v_i of the shortest vector v is upper bounded by q.*

Proof. Even though Ajtai-lattices are also infinite, the entries of the smallest vector v are in $\mathbb{Z}_q$ since $\forall x \in \mathcal{L} : x \mod q \in \mathcal{L}$ and $||x \mod q|| \leq ||x||$. □

Modifying the target function

This is a more difficult issue. Due to the way ISD works, it is restricted to finding vectors of a given (usually small) Hamming weight, and we are not able to modify it to find vectors having a small norm. However, while there is no implication between a small Hamming weight and a small norm, they are correlated. So we can use ISD to search for vectors of increasing Hamming weight and hope that the output also has a small norm.
This approach can be combined with other techniques to improve the probability that the output has small norm, e.g. to test error patterns with smaller values more often in the collision search step.

Estimating the number of non-zero entries

In code-based cryptography, the number t of non-zero values of the secret is much smaller than the length of the code. This implies that there is a relatively small number of solutions (often only one). The number of vectors over $\mathbb{F}_q$ of weight t (which usually corresponds to the size of the search space) is $\binom{n}{t}(q-1)^t$.
In lattice-based cryptography, however, the number t of non-zero values is usually unknown. In addition to that, the weight distribution can be approximated by equation (4.29) on page 92, showing that the expected weight of the solution is much larger than in the code-context. This increases the search space significantly, and also we expect a large number of vectors of a given weight, of which only few might be of small norm.

Conclusion

While ISD can be used to search for vectors in a lattice which have a small Hamming weight, this does not allow to attack lattice-based cryptosystems. The main difficulty is the different notion of *small*: the Hamming metric for codes and a norm (1-, 2-, or maximum-norm) for lattices. Due to the nature of ISD, we do not expect that this algorithm can be modified to efficiently attack lattice-based cryptosystems.

5. Parameter Selection

Our contributions

In this section, we solve the problem of selecting optimal parameters for the McEliece cryptosystem based on binary Goppa codes as well as for the QD-CFS signature scheme that provide security until a given year. This allows users to optimize the parameter choice, thereby improving the applicability of code-based cryptography. For our analysis we consider several constraints that have to be satisfied when selecting parameters:

- Security against structural attacks
- Security against decoding attacks
- Efficiency of the QD-CFS scheme (number of signing attempts and using the QD structure)

The optimal parameter set is selected from those satisfying the above constraints. The main focus is to minimize the public key size. In addition to that, we also show how to trade off signature speed with public key size for the QD-CFS scheme.

Related work

Lenstra and Verheul [50] proposed a model on how to select appropriate keys that provide security until a given year. Their work covers cryptosystems based on the factoring and the discrete logarithm, but not code-based or other post-quantum cryptosystems. As far as we know, our work is the first to apply their methodology in the context of code-based cryptography.

In an independent work Kobara [47] proposed another construction for QD code suitable for the CFS signature scheme (called flexible quasi-dyadic, or FQD) for the same problem. However, we use the construction in [8] since it has several advantages. We briefly list these below, see [8, Section 1] for details.

- The FQD construction does not produce smaller public key sizes than QD-CFS
- QD-CFS is computationally simpler
- In contrast to FQD, [8] provides a security assessment of binary QD codes against recent structural attacks [33, 94] (in particular, arguing that binary QD codes remain unscathed and are hence suitable for cryptographic applications)

Outlook and further work

As a next step, we suggest a comprehensive analysis of concrete application scenarios. In these scenarios, constraints as well as the tradeoffs between the code properties strongly depend on the details of the application, e.g. available bandwidth, acceptable response times, or typical message size. This analysis provides further insights into the current strengths and limitations of code-based cryptography, thereby also suggesting new research foci for the future.

The resulting QD-CFS scheme can be adapted to schemes derived from CFS signatures like [24], [98], or [30]. Binary QD codes can also be applied to other code-based primitives like FSB, the Stern ID scheme, or SYND. For these schemes and other applications of QD codes, optimal parameters need to be found. We leave this for further research.

5.1. Selecting Optimal Parameters

In this section, we approach the problem of selecting secure parameters for the McEliece cryptosystem based on binary Goppa codes. Our security model uses the methodology of Lenstra and Verheul (LV); in Section 5.1.3 we discuss why we chose this approach. We estimate the security of any given parameter using the lower bound complexity estimates from Section 4.1. As far as we know, this is the first work focused on this subject. Our main goal for the selection of parameters is to satisfy a minimum security level. Given that the first goal is satisfied, we want to minimize the public key size. The results of our optimization can be found in Table 5.2 on page 110.
There are, of course, other properties that can be optimized, e.g. encryption or decryption speed, or the code rate. The reason why we focus on the key sizes is the following: While the McEliece cryptosystem is already very fast and does not require special hardware (like a cryptographic coprocessor), it suffers from the drawback of having large public keys. For example, in a smart card implementation we analyzed [91], the computation time for encryption and decryption accounted for only 5% of the total time, while the data transfer of the public key (which is directly proportional to the key size) required 95% of the total time. Therefore, reducing the public key size is an important target.

5.1.1. Methodology

In this section, we present our approach to determine parameters for the McEliece cryptosystem which are secure until a given year. The analysis consists of three steps:

1. Use Lenstra and Verheul's approach to estimate the algorithmic complexity that ensures security until a given year, meaning that running a corresponding algorithm is infeasible with the available resources. This step uses a set of assumptions, e.g. about the speed of hardware development (Moore's Law)

2. Use our lower bound results (Section 4.1) for ISD, Minder and Sinclair's results [60] for GBA, and several structural attacks to estimate the security of a large number of parameters

3. Perform exhaustive search in the parameter space to select optimal parameters

Lenstra and Verheul's approach

In 1999, Lenstra and Verheul [50] described a mathematical model providing key length recommendations for public-key cryptosystems based on integer factorization and discrete logarithm. To our knowledge, this is the first important publication that uses a mathematical approach to determine secure parameters based on a set of general assumptions. After the introduction of this model, several papers made use of it to find appropriate key lengths for cryptographic primitives (see, for example, [57], [71] and [93]). Furthermore, several companies have used this model to estimate the accepted key length for their cryptographic applications. For instance, in 2004, the computer security company McAfee applied the LV-model to find the minimal key size for SSL connections [4]. Another interesting organization is the BlueKrypt company which hosts the website `www.keylength.com`. This site has an implementation of the LV-model and summarizes reports from well-known organizations, allowing the evaluation of the minimum security requirements for some symmetric and asymmetric systems in the future. In the following, we present the LV-model in more detail.

The LV-model is based on a set of assumptions that combine the impact of cryptanalytic progress and the effect of changes in computing environment. The key points of this model on which the choice of parameters depends are the following:

1. **Security margin:** It is the year s which is used to "anchor" the extrapolation. In [50] the default value of s is 1982 which represents the last year for which it is assumed that a 56-bit key DES cryptosystem provides adequate security for commercial use. The computational effort for breaking the 56-bit DES system was estimated to be $5 \cdot 10^5$ MIPS-years[1].
   In order to estimate the security level required until a given year, Lenstra and Verheul propose a function IMY(y), short for "Infeasible number of MIPS-years for year y", and it refers to the minimum computational effort that is expected to be infeasible to do in year y.
   In general, we define IMY(y) in such a way that a successful attack using tens of thousands of year-y CPUs requires more than 100 years to finish. The number of CPUs is a rough estimate for the effort a security agency might put into an attack; the number of years is derived from the fact that US law used to require some national secrets to be protected for 75 years[2], and the additional 25 years serve as a conservative buffer.

2. **Computing environment:** This estimates the changes in computational power available to attackers. The estimation is based on a slight variation of Moore's law by introducing three variables a, b, and c, which specify the changes in hardware speed, IT budget, and price over time. The definitions of these variables and their default values are as follows:

[1] MIPS = million instructions per second

[2] For example, the report on the Kennedy assassination; see `http://en.wikipedia.org/wiki/John_F._Kennedy_assassination`

- a is the average number of months in which processor speed and memory size are expected to increase by a factor of two. The default value is $a = 18$, which is the value specified by Moore's law and which is to date in line with current hardware developments. We are going to use the same value due to the fact that over the last years, hardware development has resulted in a doubling of transistors (for a fixed price) every 12–24 months[3]. Thus, a default of 18 is a compromise of this historic data. Also, opinions differ in whether hardware development will slow down due to quantum mechanical effects, or whether new technologies will further accelerate it.
- $c \in \{0, 1\}$ indicates how to interpret the variable a. For $c = 0$, the amount of computing power and memory *available to an attacker* doubles every a months, while for $c = 1$, the computing power and memory *for a given price* double every a months. We will use $c = 1$ since the historical trend mentioned above refers to a fixed price.
- b is defined as the average number of months it takes for IT budgets to double. It is assumed that the budget available to an attacker (be it a single person or a security agency) develops similarly to the overall economy. According to historic data[4], the US Gross National Product has doubled approx. every 10.5 years over the last 30 years. Since the exact growth varies every year, we will use an average value to extrapolate over a larger period of time. Our default setting for b is 120.

3. **Cryptanalysis:** This refers to the future cryptanalytic progress. It is measured by the number of months r it is expected for cryptanalytic attacks to become twice as efficient, i.e. to require half the number of operations for a comparable task. We estimate this number extrapolating the efficiency of attacks against code-based cryptosystems only, since the cryptanalytic development can be very different for other cryptosystems. Lenstra and Verheul's default value is $r = 18$. In code-based cryptography, we find it reasonable to assume that the pace of future cryptanalytic developments and their impact will be relatively close to historic developments from 1988 until 2009. By applying a linear regression on data points listed in Table 2.2, we see a twofold attack efficiency improvement every $2,44$ years. We use $r = 30$, which corresponds to 2.5 years.

At the end of this section, we will provide a sensitivity analysis for the variables in our model.

Based on these parameters, Lenstra and Verheul present a formula which can be used to derive lower bounds for the algorithmic complexity that offers a specified security level at least until year y in the future (independent of the concrete asymmetric cryptosystem).

[3]See http://wi-fizzle.com/compsci/
[4]See http://www.bea.gov

Corollary 5.1.1. *Under the assumptions described above regarding the complexity of breaking DES and the hardware and budget developments in the future, the number of MIPS-years that are expected to remain infeasible at least until year y is*

$$IMY(y) = 5 \cdot 10^5 \cdot 2^{12(y-s)/a} \cdot 2^{12c(y-s)/b} \quad \textit{MIPS-years.} \tag{5.1}$$

With our default settings, it follows that in year y, a computational complexity of

$$\mathrm{IMY}(y) = 5 \cdot 10^5 \cdot 2^{\frac{23}{30}(y-1982)} \quad \text{MIPS-years} \tag{5.2}$$

provides an acceptable level of security. The next step is to convert this lower bound expressed in MIPS-years to a lower bound for the number of binary operations.

Corollary 5.1.2. *Using as a data point the result [15] that in 2008, approximately $2^{60.4}$ binary operations were needed to decode a message which was encoded using the original McEliece parameters $(1024, 524, 50)$, and expecting cryptanalytic developments by a factor $2^{12(y-2008)/r}$ (with $r = 30$); a sufficient condition for the security level of a McEliece instance with parameter set (n, k, t), denoted $S(n, k, t)$, providing an adequate security until a given year y is the following:*

$$S(n, k, t) \geq \frac{IMY(y) \cdot 2^{12(y-2008)/30} \cdot 2^{60.4}}{1.7 \cdot 10^5}. \tag{5.3}$$

As in [50, Page 9], the value of $S(n, k, t)$ is defined as the expected runtime of the fastest algorithm published in 2008 for attacking the McEliece cryptosystem with the parameter set (n, k, t). In our case, this corresponds to the lower bounds presented in [67, 68]. The value $1.7 \cdot 10^5$ is expressed in MIPS-years and obtained from the fact that the attack by Bernstein et al. [15] required 1400 CPU days on Q6600 quad processors. Assuming that a Q6600 processor executes approximately $44,000$ MIPS (SiSoft Sandra benchmark and [1]), this corresponds to $1.7 \cdot 10^5$ MIPS-years.
Therefore, the inequality (5.3) becomes:

$$S(n, k, t) \geq 2.9412 \cdot 2^{\frac{23}{30}(y-1982)+\frac{12}{30}(y-2008)+60.4} \tag{5.4}$$

5.1.2. Sensitivity analysis

In this section, we analyze the choice of the values we described in the previous Section 5.1.1. More specifically, we estimate the impact that a different value of each variable has on the resulting security level. This analysis quantifies the robustness of our model, and it allows users to apply our results even if they have different assumptions about the correct values.

1982 and DES-56 bit

The function $\mathrm{IMY}(y)$ was "anchored" by using 1982 as the last year in which breaking the DES scheme with 56-bit key was considered infeasible. The choice to use DES with 56-bit for this definition is arbitrary; the function, therefore, is defined using the number of operations required to break the DES scheme, and it is thus independent of which

cryptosystem was used for the definition. Any other year and/or cryptosystem can be used for the definition, e.g. AES or RSA. Using the data from the recent attack by Bernstein et al. [15] that a 44,000 MIPS CPU breaks the original McEliece parameters in 1400 CPU-days, the attack complexity estimated as 2^{60} operations corresponds to $2^{17.3}$ MIPS years. An attack complexity of 2^{80} operations, which is considered the "smallest general-purpose level" of security[5], corresponds to $2^{37.3}$ MIPS-years, very close to our estimate of IMY(2008) = $2^{38.8}$.

Moore's Law (parameters a and c)

The original Moore's Law refers to the number of transistors on an integrated circuit [62]. Moore estimated this number to double every two years[6]. The number of MIPS of a CPU depends on the number of transistors, but also on the clock speed. These two factors taken together increase the chip performance by a factor of two every 18 months (estimated by David House, an Intel executive). For our sensitivity analysis, we will consider a 10% error in this estimate, i.e. a range between 16 and 20 months for a twofold performance increase. The value $c = 1$ is in line with past developments, but we will show the impact of $c = 0$ below.

Budget (parameter b)

Our choice for the value of b is based on the budget development of the US, since they constitute the largest economic power worldwide. However, countries like China have a much higher economic growth; some analysts expect China to overtake the US in the near future, doubling the US economic power in 2030[7]. This growth corresponds to a twicefold increase in economic power in 6 years. Even though the GDP of China is smaller than that of the US (about 40% in 2010) and the faster growth is therefore on a smaller baseline, we will assume a range of 72–120 for the value of b.

Cryptanalytic progress (parameter r)

For more than two decades, cryptanalytic progress has improved the efficiency of the fastest attack algorithm by a factor of two every 30 months. While every individual attack algorithm has a lower bound for its complexity (see, for example, [15, 36, 67]), many new attacks have been developed which improved the previous bounds. As in the case of Moore's Law, it is unclear whether generic attack algorithms have a lower bound for their complexity that cannot be improved, thereby slowing down cryptanalytic progress, or whether new cryptanalytic tools will increase the progress. We will therefore consider a larger range for r, from 20 to 40 months.

[5] www.keylength.com, ECRYPT II recommendations
[6] See `http://www.intel.com/technology/mooreslaw/` or `http://en.wikipedia.org/wiki/Moore's_law`
[7] J. Lin, World Bank's chief economist, on March 23rd, 2011

Table 5.1.: Impact of different input values to our model. Impact is the absolute and relative change in the required security level for the year 2050.

Parameter	Our value	Expected range	Impact	
			Absolute	Relative
a	18	16–20	5.7	4.3%
b	120	72–120	4.5	3.5%
c	1	0/1	6.8	5.2%
r	30	20–40	8.4	6.4%

Conclusion

It can be seen from Table 5.1 that even very pessimistic assumptions (from a user's point of view) do not lead to dramatic changes in the required security level. For example, assuming the most pessimistic value for all four parameters above raises the required security for 2050 from 131 to 149 bit, an increase of 18.6 bit or 14.2%.
Therefore, if a user has different assumptions than we used to compute Table 5.2, appropriate parameters can be found as follows: Determine the necessary security level using our table and the year y until which protection is required; increase this security level using the values in Table 5.1 (or correspondingly smaller values if $y < 2050$); locate the row in Table 5.2 that corresponds to this higher security level and find optimal parameters in it. Table 5.3 (page 112) applies the optimistic and pessimistic assumptions described above and shows optimal parameters for selected years.

5.1.3. Discussion on the choice of the LV methodology

While we gave several examples above where the LV methodology was used, we acknowledge that it is not widely applied in the academic world nor outside of it. There are other methods of choosing parameters, e.g. one set of parameters for each output length of the employed hash function. These alternatives are acceptable in the academic context where the focus of the work is not on the parameters, but on the efficiency improvement of the algorithm, for instance. However, we argue that the LV methodology is superior, especially outside the academic world, where the parameter choice is important for cryptographic applications.

The basic question when choosing parameters is: "which security level is required to withstand attacks until a given year in the future?". This level is then translated into parameters in a way depending on the application. Many alternative methods like the one mentioned above do not answer this question, and instead of choosing parameters, a user faces the equally difficult question of choosing a hash function length.
In order to answer this question, one has to model the development of "available attack power" which the given application will face. This can be done in very different ways, and in different units of measurement, where MIPS-years and binary operations are two prominent examples.
The LV methodology is a very natural framework to model this development. It is based

on a clear logic tree of drivers: available attack power increases through algorithmic and hardware improvements, hardware development breaks down into computing power per dollar and budget increases etc. This logic not only reflects the way we naturally think about this problem, it has several advantages:

- Its structure and level of detail make the results credible and convincing
- It makes assumptions transparent and allows to identify the biggest drivers
- It allows sensitivity analyses to test and validate the assumptions and, depending on the results, potentially adjust them

For these reasons, we chose the LV framework to select optimal parameters.

5.1.4. Structural attacks against the Goppa code structure

As pointed out in [28], the only structural attack against general Goppa codes is the support splitting algorithm (SSA) [81]. A brief description of the algorithm can be found in Section 2.5.2. We will provide a rough estimate of its complexity; in the next section, we will check all parameters against this attack to ensure that they are not vulnerable against it (meaning that the SSA has a higher complexity than ISD or GBA.

Given a public key corresponding to the McEliece cryptosystem, the SSA does not directly recover the secret key. Instead, the attacker has to iterate through all possible secret keys and check whether the corresponding Goppa code is permutation equivalent to the public key. The complexity of a SSA-based attack against an (n, k, t) Goppa code is approximately

$$tmn^{t-2}$$

binary operations, where $m = \lceil \log_2 n \rceil$.

Sendrier showed that the complexity of one run of the SSA against a code $\mathcal{C}$ is in $\mathcal{O}(2^{\dim(\mathcal{C} \cap \mathcal{C}^\perp)})$. The set $\mathcal{C} \cap \mathcal{C}^\perp$ is called the hull of the code $\mathcal{C}$. Since the hull of a code is often of small dimension, the complexity of the algorithm is not very high. However, it cannot be smaller than $n(tm)^2$, where $m = \lceil \log_2 n \rceil$, which corresponds to a Gaussian elimination. The number of codes that need to be tested can be computed using [80]. As an approximation, there are $2^{tm}/t$ binary $(n = 2^m, k, t)$ Goppa codes, and one out of mn^3 need to be tested, so we have an approximate complexity of this attack of tmn^{t-2} binary operations (see [28]).

5.1.5. Optimal parameters

The problem of estimating secure parameters for the McEliece cryptosystem for a given year consists in obtaining, for the security level S calculated in equation (5.3), a set of parameters that achieves this security level and provides the smallest key size among all other such sets. To solve this problem, we first show how an instance attaining maximum security for a given key size can be used to solve the problem of finding the optimal key

size for a given security level. We then present an algorithm that we use to find optimal instances.

From [69] we know that there exists an optimal information rate $R^* = k/n$ with $R^* \approx 0.8$ such that, for a given key size, the highest security is achieved at this rate. Although this has been shown using a generic estimate for the security of given parameter sets and we use more advanced lower bounds from [36, 67], we can expect that for a given key size K the maximum security will be achieved at some R^*, which is similar for all K. This is mainly due to the fact that the improvements of the ISD algorithm do not seem to improve much on the exponential part but on the polynomial factor. Moreover, because of the same reason we expect this R^* not to differ significantly from the value 0.8 predicted by [69, Lemma 1].
Building on this, we construct an algorithm that finds, with arbitrary precision, an instance with the smallest key size possible achieving the given security level S. This algorithm is depicted below (see Algorithm 16). In this algorithm, the value of S is calculated via the inequality (5.4), and the interval $[R_{start}, R_{end}]$ is chosen large enough and contains 0.8: we take an information rate which ranges from $R_{start} = 0.6$ to $R_{end} = 0.85$. All other parameters are selected such that it is feasible to complete the algorithm in a reasonable time. For the key size, we set $K_{up} = 200$ kB as an upper bound and use the step size $K_{step} = 1$ kB. Moreover, we use the lower bound formula from [67] as a function C.

Our results are presented in Table 5.2 which shows the following information:

- Year: the year until which data security is required. Historic data is given mainly to allow comparison with other sources.

- Symmetric key size: the symmetric key size required to ensure data security, calculated in accordance with Lenstra and Verheul's approach.

- Lower bound for $\log_2(S(n, k, t))$: the $\log_2$ of the minimum number of binary operations (required to break a McEliece cryptosystem) that are infeasible in the respective year.

- The last two columns are a translation of the required symmetric key size into parameters relevant in practice, i.e. the number of MIPS years that render a cryptosystem infeasible to break, and the corresponding number of years on a modern quad core CPU.

5.2. Quasi-dyadic CFS signatures

In this section, we contribute results to three questions regarding the QD-CFS signature scheme. We will provide a security analysis regarding structural as well as decoding attacks, and propose parameters based on the methodology presented in the previous Section 5.1. In addition to that, we show how the parameters can be used to trade off signature generation time versus public key size.

Table 5.2.: Proposed parameters for the McEliece cryptosystem — optimized for public key size

Year	Symmetric Key Size	Lower bound for $\log_2 S(n,k,t)$	McEliece parameters (n,k,t) and public key size (kB)		Infeasible number of MIPS-years	Corresponding number of years[a]
2009	77	83	(1641, 1213, 40)	63	$8.52 \cdot 10^{11}$	$1.94 \cdot 10^{7}$
2010	78	84	(1661, 1222, 41)	66	$1.45 \cdot 10^{12}$	$3.30 \cdot 10^{7}$
2011	79	85	(1680, 1230, 42)	68	$2.47 \cdot 10^{12}$	$5.61 \cdot 10^{7}$
2012	80	87	(1722, 1270, 42)	70	$4.19 \cdot 10^{12}$	$9.52 \cdot 10^{7}$
2013	80	88	(1740, 1277, 43)	72	$7.14 \cdot 10^{12}$	$1.62 \cdot 10^{8}$
2014	81	89	(1765, 1290, 44)	75	$1.21 \cdot 10^{13}$	$2.75 \cdot 10^{8}$
2015	82	90	(1801, 1325, 44)	77	$2.07 \cdot 10^{13}$	$4.70 \cdot 10^{8}$
2016	83	91	(1859, 1392, 43)	79	$3.51 \cdot 10^{13}$	$7.98 \cdot 10^{8}$
2017	83	92	(1861, 1372, 45)	82	$5.98 \cdot 10^{13}$	$1.36 \cdot 10^{9}$
2018	84	93	(1922, 1442, 44)	85	$1.02 \cdot 10^{14}$	$2.32 \cdot 10^{9}$
2019	85	95	(1923, 1421, 46)	87	$1.73 \cdot 10^{14}$	$3.93 \cdot 10^{9}$
2020	86	96	(1983, 1490, 45)	90	$2.94 \cdot 10^{14}$	$6.68 \cdot 10^{9}$
2021	86	97	(1983, 1468, 47)	92	$5.01 \cdot 10^{14}$	$1.14 \cdot 10^{10}$
2022	87	98	(2044, 1538, 46)	95	$8.52 \cdot 10^{14}$	$1.94 \cdot 10^{10}$
2023	88	99	(2045, 1517, 48)	98	$1.45 \cdot 10^{15}$	$3.30 \cdot 10^{10}$
2024	89	101	(2072, 1532, 49)	101	$2.47 \cdot 10^{15}$	$5.61 \cdot 10^{10}$
2025	89	102	(2135, 1604, 48)	104	$4.20 \cdot 10^{15}$	$9.55 \cdot 10^{10}$
2026	90	103	(2157, 1614, 49)	107	$7.14 \cdot 10^{15}$	$1.62 \cdot 10^{11}$
2027	91	104	(2159, 1594, 51)	110	$1.21 \cdot 10^{16}$	$2.75 \cdot 10^{11}$
2028	92	105	(2187, 1621, 51)	112	$2.07 \cdot 10^{16}$	$4.70 \cdot 10^{11}$
2029	93	106	(2241, 1673, 51)	116	$3.52 \cdot 10^{16}$	$8.00 \cdot 10^{11}$
2030	93	108	(2248, 1669, 52)	118	$5.98 \cdot 10^{16}$	$1.36 \cdot 10^{12}$
2032	95	110	(2340, 1758, 52)	125	$1.73 \cdot 10^{17}$	$3.93 \cdot 10^{12}$
2034	96	112	(2396, 1801, 53)	131	$5.01 \cdot 10^{17}$	$1.14 \cdot 10^{13}$
2036	98	115	(2443, 1824, 55)	138	$1.45 \cdot 10^{18}$	$3.30 \cdot 10^{13}$
2038	99	117	(2531, 1909, 55)	145	$4.20 \cdot 10^{18}$	$9.55 \cdot 10^{13}$
2040	101	119	(2573, 1927, 57)	152	$1.22 \cdot 10^{19}$	$2.77 \cdot 10^{14}$
2042	103	122	(2614, 1944, 59)	159	$3.52 \cdot 10^{19}$	$8.00 \cdot 10^{14}$
2044	104	124	(2653, 1959, 61)	166	$1.02 \cdot 10^{20}$	$2.32 \cdot 10^{15}$
2046	106	126	(2774, 2099, 59)	173	$2.95 \cdot 10^{20}$	$6.70 \cdot 10^{15}$
2048	107	129	(2798, 2088, 62)	181	$8.53 \cdot 10^{20}$	$1.94 \cdot 10^{16}$
2050	109	131	(2804, 2048, 66)	189	$2.47 \cdot 10^{21}$	$5.61 \cdot 10^{16}$

[a] on a 2.4 GHz Intel Core 2 Quad Q6600

Algorithm 16 Search$(S, C, K_{step}, K_{up}, R_{step}, R_{start}, R_{end})$

Require:
- Security level S
- Complexity function $C(n, R)$
- Step for the key size search K_{step}
- Search upper bound for the key size K_{up}
- Step for the rate search R_{step}
- Rate search interval bounds R_{start}, R_{end}

Ensure: n_{out} and R_{out} such that
- $C(n_{out}, R_{out}) \geq S$
- The key size is the smallest possible up to steps K_{step} and R_{step}

Begin
for $K = K_{step}$ to K_{up} **do**
  for $R = R_{start}$ to R_{end} **do**
    $n \leftarrow \sqrt{\frac{K}{R(1-R)}}$
    if $C(n, R) \geq S$ **then**
      return n and R
    end if
    $R \leftarrow R + R_{step}$
  end for
  $K \leftarrow K + K_{step}$
end for
return "NO solution found"
End

5.2.1. Structural attacks against the QD structure

In addition to structural attacks against the Goppa structure (which we have covered in Section 5.1.4), we also have to consider structural attacks against the QD structure when using QD-CFS. In 2010, Faugère et al. [33] published a structural attack against the McEliece cryptosystem when used with QC or QD codes. Since a dual of McEliece is the Niederreiter cryptosystem, on which the CFS signature scheme is based, this attack has to be taken into account in our context.

Proposition 5.2.1. *A necessary condition to protect against the structural attack by Faugère et al. is that non-binary QD Goppa codes have to be avoided and that the complexity of guessing half the errors needs to be large enough to be infeasible. The latter constraint is met if* $\binom{n}{t/2}$ *is greater than the complexity which is considered infeasible (computed as in Section 5.1.1).*

In order to show why Proposition 5.2.1 is true, we will briefly describe their attack here and then discuss how to protect against it.

Table 5.3.: Comparison of parameters using optimistic versus pessimistic assumptions (from a users point of view) for selected years.

	Optimistic scenario		Pessimistic scenario	
Year	Lower bound for $\log_2 S(n,k,t)$	McEliece parameters (n,k,t) and public key size (kB)	Lower bound for $\log_2 S(n,k,t)$	McEliece parameters (n,k,t) and public key size (kB)
2020	95	(1902, 1390, 47) 87	98	(2047, 1541, 46) 95
2030	105	(2220, 1664, 50) 113	112	(2396, 1801, 53) 131
2040	142	(2453, 1811, 57) 116	126	(2730, 2045, 60) 171
2050	127	(2732, 2024, 62) 175	139	(3108, 2342, 66) 219

The first step is to recognize that Goppa codes are a subclass of alternant codes, so any QD Goppa code is also an alternant code. Alternant codes can be defined using parity check matrices of a certain structure. We define the alternant code $\mathcal{A}_r(x,y)$ over $\mathbb{F}_q$ as

$$\mathcal{A}_r(x,y) = \{v \in \mathbb{F}_q^n : V_r(x,y)v^{\mathrm{T}} = 0\},$$

where $x = (x_0, \ldots, x_{n-1}) \in \mathbb{F}_{q^m}^n$ is a vector of pairwise different elements, $y = (y_0, \ldots, y_{n-1}) \in \mathbb{F}_{q^m}^n$ a vector of non-zero elements, $m = \lceil \log_2(n) \rceil$, and

$$V_r(x,y) = \begin{pmatrix} y_0 & \cdots & y_{n-1} \\ y_0 x_0 & \cdots & y_{n-1} x_{n-1} \\ \vdots & & \vdots \\ y_0 x_0^{r-1} & \cdots & y_{n-1} x_{n-1}^{r-1} \end{pmatrix}.$$

Secondly, there exist polynomial time algorithms for decoding alternant codes; such an algorithm can be used once a parity check matrix in the form $H = V_r(x,y)$ for the code is found. Let $\mathcal{C}$ be the code on which a QD-CFS scheme is based, G the corresponding public generator matrix (which can be derived from the public parity check matrix, if necessary). For any parity check matrix H of $\mathcal{C}$, we have $GH^{\mathrm{T}} = 0$.
Since the secret matrix H defines a Goppa code, the linear equation system corresponding to 5.2.1 has the following form:

$$\{g_{1,1}Y_1X_1^j + \cdots + g_{1,n}Y_nX_n^j = 0 : 1 \le i \le k, 0 \le j \le r-1\},$$

where $g_{i,j}$ are the (known) entries of G and X and Y the unknowns. Usually, this system is too complex to be solved. However, using the QD structure allows to greatly reduce the number of unknowns and therefore solve the system. The solutions yields an alternant decoder, i.e. an algorithm for decoding alternant codes; for codes over non-binary fields $\mathbb{F}_q$, this decoder allows an attacker to forge signatures.

Binary Goppa codes, however, have approximately twice the minimum distance than corresponding binary alternant codes: For a code of length n and dimension k, alternant codes have a minimum distance

$$\geq \frac{n-k}{\lceil \log_2(n) \rceil} + 1,$$

whereas Goppa codes have a minimum distance of

$$\geq 2\frac{n-k}{\lceil \log_2(n) \rceil} + 1.$$

Therefore, a binary alternant decoder allows to decode only half the errors that can be added to a codeword of a binary Goppa code — the other half an attacker needs to guess. Proposition 5.2.1 ensures that this type of attack remains infeasible.

Remark 5.2.2. *As pointed out in Proposition 5.2.1, the conditions above are* necessary, *not* sufficient *to protect against the attack by Faugère et al. While no efficient algorithm is known to extend the $t/2$-error correcting alternant decoder into a t-error correcting Goppa decoder, we cannot yet exclude the possibility that such an algorithm exists. In this case, a parameter adjustment would be necessary for QD Goppa codes to remain secure.*

5.2.2. Decoding attacks against the QD structure

As described in Section 2.5.2, (QD-)CFS signatures constitute a "decoding one out-of-many" (DOOM) scenario since an attacker can be satisfied with decoding any of the syndromes generated by hashing the document with different counters. However, we do not suggest to apply Finiasz' "Parallel-CFS" [35] construction mentioned in [83]: it requires complete decoding which is very inefficient, and it is unclear whether this construction is really immune to the above attack.
Thus, we propose parameters which remain secure even facing a decoding one out-of-many attack by increasing the required security level by 50% (since a DOOM attack reduces the runtime exponent to approximately 2/3).

5.2.3. Optimal QD-CFS parameters

When selecting parameters for the QD-CFS signature scheme, there are several constraints that need or should be taken into account:

- Security:
  - The complexity of ISD and GBA attacks need to be large enough, in order to protect against these generic attacks
  - To protect against the structural attack discussed in Section 5.2.1, avoid using non-binary QD Goppa codes, and choose parameters such that guessing half the errors is infeasible
  - The number of non-equivalent Goppa codes of corresponding parameters needs to be large enough to render SSA-based attacks infeasible
- Efficiency:
  - To utilize the QD structure, t should be a multiple of a large power of 2
  - Use codes of high density to keep the number of signing attempts small
  - The parameters can be used to trade off speed versus key size, depending on the application

Table 5.4.: Proposed parameters for the QD-CFS signature scheme — optimized for public key size and number of signing attempts.

Year	Symmetric Key Size	Lower bound for $\log_2 S(n,k,t)$ (incl. DOOM)	McEliece parameters (n,k,t) and public key size (MB)	
2014	81	122	(1979448, 1979196, 12)	14.8
2020	86	129	(3958896, 3958632, 12)	31.1
2030	93	140	(15835583, 15835295, 12)	135.9
2040	101	152	(31671167, 31670867, 12)	283.2
2050	109	164	(126684666, 126684342, 12)	1223.3

Table 5.5.: Minimum value of m to yield a time complexity of at least 2^{80} (incl. DOOM attacks); corresponding expected number of signing attempts and key sizes.

(t,m)	$(8,29)$	$(10,24)$	$(12,21)$	$(14,19)$
Security level	$2^{81.6}$	$2^{81.8}$	$2^{81.3}$	$2^{83.9}$
Avg no. of sign attempts	$2^{16.3}$	$2^{22.8}$	$2^{29.8}$	$2^{37.3}$
Key size (MB)	822	224	15	8

The protection against the attack by Faugère et al. and against the SSA has been discussed in Sections 5.2.1 and 5.1.4. In the following, we propose parameters that are secure against ISD and GBA attacks and satisfy the necessary conditions to be secure against those two structural attacks. In order to do this, we make use of the methodology introduced in Section 5.1. Among those parameters sets that satisfy the security constraints, we will select the one minimizing the key size (taking into account the effect of the QD structure). The results can be found in Table 5.4, followed by a discussion on the tradeoff between speed and key size.

As we described in Section 2.4.3, the CFS signature scheme requires a much smaller value of t in order to reduce the number of signing attempts, which is in $\mathcal{O}(t!)$. However, this requires large values of $n \leq 2^m$ and k, which increases the public key size pk, where $\text{pk} = k(n-k)\lceil \log_2 q \rceil$ bits. Hence, the (QD-)CFS signature scheme allows to trade off signing speed (in number of signing attempts) against public key size. An example is given in Table 5.5.

For the security range we considered in Table 5.4, a value of $t = 12$ has shown to be optimal. Smaller values lead to larger key sizes since 12 is dividable by a relatively large power of 2 and therefore allows to use the QD structure efficiently. The choice of $t > 12$ is suboptimal in our view, since the relatively small key size reduction is not outweighed by the exponential increase in the number of signing attempts.
In addition to that, we propose to use the greatest possible code length $n = \lfloor 2^{m-1/t} \rceil$. While smaller values of n slightly reduce the public key size, they significantly increase the number of signing attempts: If the code length is a fraction $1/2^c$ of the maximal length 2^m, the number of signing attempts is increased by a factor of 2^{ct} (see [8] for more details).

6. Conclusion and further research

In this thesis, we presented contributions to the theoretical security as well as the practical application of code-based cryptography.
We started with a study of attacks in a broadcast scenario. To our knowledge, this type of attack has not been analyzed in the context of code-based cryptography. In this section, we found that the Niederreiter and the HyMES cryptosystems are vulnerable to a broadcast attack, in contrast to the McEliece encryption scheme. In addition to that, we generalized the broadcast attack to the situation of similar (instead of identical) messages, and we proved bounds for the expected number of recipients that are required to run our attack in both cases. If an attacker intercepts a smaller number of messages, it is possible to use the gathered knowledge to improve an ISD attack, and we computed the respective attack complexity. A discussion on the use of semantic conversions to protect against broadcast attacks concluded our analysis, and we implemented our attack in Java to demonstrate its efficiency.
The subsequent section of this thesis covers our improvements of non-critical attacks. Like many other attacks, ISD had largely been studied in the context of binary codes. We generalized this attack to codes of larger fields $\mathbb{F}_q$, developed an additional efficiency improvement, and proved lower bounds for the complexity. The next result is a study of the effect of partial knowledge on the efficiency of ISD attacks. After defining two types of partial knowledge and describing situations where an attacker might be able to obtain this knowledge, we presented techniques to exploit this knowledge and proved lower bounds for the complexity of ISD attacks modified respectively. Recognizing the number of publications which make use of highly structured matrices, we developed a modified GBA attack that makes use of this structure to improve its efficiency. This results demonstrates that additional code structure — aimed to achieve a smaller public key size or other improvements — potentially decreases the security of a given cryptosystem. The subsequent contribution is the generalization of statistical decoding from binary to larger finite field $\mathbb{F}_q$. We provided a theoretical analysis of the success probability of our algorithm, and gathered experimental data for different field sizes. Many instances were successfully decoded, independent of the field size. In addition to that, we described techniques that can be used to exploit partial knowledge in order to increase the algorithm's efficiency. Our final result in this section is a study of two lattice attacks against code-based cryptosystems, and of ISD against lattice-based schemes.
Our final results concern the selection of appropriate parameters. We applied the Lenstra-Verheul framework to the McEliece encryption scheme and the QD-CFS signature scheme to select secure and efficient parameters. In order to do this, we discussed in detail our assumptions regarding future hardware and software developments on which this framework is built, and added a sensitivity analysis to show the robustness of our results.

Bibliography

[1] Intel Core 2 Quad Q6600, May 2007. Cited on page 105.

[2] A. May A. Meurer A. Becker, A. Joux. Decoding random binary linear codes in $2^{(n/20)}$: How $1 + 1 = 0$ improves information set decoding. To appear in Advances in Cryptology (Eurocrypt 2012), 2012. Cited on page 60.

[3] C.M. Adams and H. Meijer. Security-related comments regarding McEliece public-key cryptosystem. *IEEE Transactions on Information Theory*, 35(2):454–455, 1989. Cited on page 37.

[4] R. Araujo. The need for strong SSL ciphers, 2004. Cited on page 103.

[5] D. Augot, M. Finiasz, and N. Sendrier. A family of fast syndrome based cryptographic hash functions. In Ed Dawson and Serge Vaudenay, editors, *Mycrypt*, volume 3715 of *Lecture Notes in Computer Science*, pages 64–83. Springer, 2005. Cited on pages 19, 31, 36, 61, 62, and 79.

[6] A. Barg. Some new NP-complete coding problems. *Probl. Peredachi Inf.*, 30:23–28, 1994. (in Russian). Cited on page 26.

[7] A. Barg. Complexity issues in coding theory. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(46):1–115, 1997. Cited on pages 23 and 26.

[8] P. S. L. M. Barreto, P.-L. Cayrel, R. Misoczki, and R. Niebuhr. Quasi-dyadic CFS signatures. In *Inscrypt 2010*, volume 6584 of *Lecture Notes in Computer Science*. Springer, 2010. Cited on pages 62, 79, 83, 101, and 114.

[9] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *EUROCRYPT*, pages 92–111, 1994. Cited on page 54.

[10] T. P. Berger, P.-L. Cayrel, P. Gaborit, and A. Otmani. Reducing key length of the McEliece cryptosystem. In *Progress in Cryptology – Africacrypt'2009*, volume 5580 of *Lecture Notes in Computer Science*, pages 77–97. Springer, 2009. Cited on pages 20, 24, 25, 62, 69, 79, and 85.

[11] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(2):384–386, May 1978. Cited on page 26.

[12] D. J. Bernstein. Grover vs. McEliece. In N. Sendrier, editor, *PQCrypto*, volume 6061 of *Lecture Notes in Computer Science*, pages 73–80. Springer, 2010. Cited on page 62.

[13] D. J. Bernstein, J. Buchmann, and E. Dahmen. *Post-Quantum Cryptography*. Springer, 2008. Cited on pages 32, 43, 44, 45, 55, and 60.

[14] D. J. Bernstein, T. Lange, R. Niederhagen, C. Peters, and P. Schwabe. FSBday. In *INDOCRYPT*, pages 18–38, 2009. Cited on pages 39, 61, and 81.

[15] D. J. Bernstein, T. Lange, and C. Peters. Attacking and defending the McEliece cryptosystem. In *PQCrypto '08: Proceedings of the 2nd International Workshop on Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 31–46. Springer, 2008. Cited on pages 37, 57, 60, 105, and 106.

[16] D. J. Bernstein, T. Lange, and C. Peters. Smaller decoding exponents: Ball-collision decoding. In *CRYPTO*, pages 743–760, 2011. Cited on pages 37, 57, 58, 60, 63, 69, and 85.

[17] D. J. Bernstein, T. Lange, C. Peters, and P. Schwabe. Faster 2-Regular Information-Set Decoding. In Y. M. Chee, Z. Guo, S. Ling, F. Shao, Y. Tang, H. Wang, and C. Xing, editors, *IWCC*, volume 6639 of *Lecture Notes in Computer Science*, pages 81–98. Springer, 2011. Cited on pages 32 and 61.

[18] B. Biswas. *Implementational aspects of code-based cryptography*. PhD thesis, École Polytechnique, Paris, France, 2010. Cited on pages 43 and 44.

[19] G. Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990. Cited on pages 119 and 121.

[20] J. Buchmann. FlexiProvider, May 2011. Developed by the Theoretical Computer Science Research Group of Prof. Dr. Johannes Buchmann at the Departement of Computer Science at Technische Universität Darmstadt, Germany. Cited on page 54.

[21] A. Canteaut and H. Chabanne. A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem. Research Report RR-2227, INRIA, 1994. Cited on page 37.

[22] A. Canteaut and F. Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998. Cited on pages 37 and 60.

[23] A. Canteaut and N. Sendrier. Cryptoanalysis of the original McEliece cryptosystem. In K. Ohta and D. Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 1998. Cited on page 34.

[24] P.-L. Cayrel, P. Gaborit, D. Galindo, and M. Girault. Improved identity-based identification using correcting codes. *CoRR*, abs/0903.0069, 2009. Cited on page 102.

[25] P.-L. Cayrel, P. Gaborit, and M. Girault. Identity-based identification and signature schemes using correcting codes. 2007. Cited on page 20.

[26] P.-L. Cayrel, P. Véron, and S. M. El Yousfi Alaoui. Improved code-based identification scheme. Lecture Notes in Computer Science. Springer, 2010. `http://arxiv.org/abs/1001.3017v1`. Cited on pages 58, 62, 77, and 78.

[27] K.-M. Cheung. The weight distribution and randomness of linear codes. TDA Progress Report 42–97, Communications Systems Research Section, 1989. Cited on page 90.

[28] N. Courtois, M. Finiasz, and N. Sendrier. How to achieve a McEliece-based digital signature scheme. In *Advances in Cryptology – Asiacrypt'2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 157–174, Gold Coast, Australia, 2001. Springer. Cited on pages 28, 36, 38, and 108.

[29] L. Dallot. Towards a concrete security proof of Courtois, Finiasz and Sendrier signature scheme. In S. Lucks, A.-R. Sadeghi, and C. Wolf, editors, *WEWoRC*, volume 4945 of *Lecture Notes in Computer Science*, pages 65–77. Springer, 2007. Cited on pages 26 and 30.

[30] L. Dallot and D. Vergnaud. Provably secure code-based threshold ring signatures. In Matthew G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2009. Cited on pages 20 and 102.

[31] I. Damgård. A design principle for hash functions. In Brassard [19], pages 416–427. Cited on page 31.

[32] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. A Distinguisher for High Rate McEliece Cryptosystems. eprint Report 2010/331, 2010. Cited on pages 26 and 51.

[33] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic Cryptanalysis of McEliece variants with compact keys – toward a complexity analysis. In *SCC '10: Proceedings of the 2nd International Conference on Symbolic Computation and Cryptography*, pages 45–55, RHUL, June 2010. Cited on pages 19, 62, 101, and 111.

[34] M. Finiasz. NP-completeness of certain sub-classes of the syndrome decoding problem. *CoRR*, abs/0912.0453, 2009. Cited on page 26.

[35] M. Finiasz. Parallel-cfs - strengthening the cfs mceliece-based signature scheme. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2010. Cited on page 113.

[36] M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In M. Matsui, editor, *Advances in Cryptology – Asiacrypt'2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009. Cited on pages 37, 38, 57, 58, 60, 63, 64, 65, 68, 69, 106, 109, and 130.

[37] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO*, pages 537–554, 1999. Cited on page 54.

[38] P. Gaborit and M. Girault. Lightweight code-based authentication and signature. In *IEEE International Symposium on Information Theory – ISIT'2007*, pages 191–195, Nice, France, 2007. IEEE. Cited on page 85.

[39] P. Gaborit, C. Laudaroux, and N. Sendrier. SYND: a very fast code-based cipher stream with a security reduction. In *IEEE Conference, ISIT'07*, pages 186–190, Nice, France, Jul 2007. Cited on pages 19 and 85.

[40] R. G. Gallager. *Low-Density Parity-Check Codes*. PhD thesis, 1963. Cited on page 20.

[41] N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In *Advances in Cryptology – Proceedings of EUROCRYPT '10*, volume 6110 of *Lecture Notes in Computer Science*. Springer, 2010. Cited on pages 59 and 97.

[42] E. N. Gilbert. A comparison of signalling alphabets. *The Bell system technical journal*, 31:504–522, 1952. Cited on page 23.

[43] J. Håstad. Solving simultaneous modular equations of low degree. *SIAM J. Comput.*, 17(2):336–341, 1988. Cited on page 44.

[44] I. F. Blake (ed.). *Algebraic coding theory: history and development*. Dowden, Hutchinson & Ross, 1973. Cited on page 124.

[45] A. K. Al Jabri. A statistical decoding algorithm for general linear block codes. In B. Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 1–8. Springer, 2001. Cited on pages 61 and 85.

[46] P. Kirchner. Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377, 2011. `http://eprint.iacr.org/`. Cited on page 61.

[47] K. Kobara. Flexible quasi-dyadic code-based public-key encryption and signature. Cryptology ePrint Archive, Report 2009/635, 2009. Cited on page 101.

[48] K. Kobara and H. Imai. Semantically secure McEliece public-key cryptosystems - conversions for McEliece PKC. In *In Proceedings of 4th International Workshop on Practice and Theory in Public Key Cryptosystems (PKC '01)*, pages 19–35, 2001. Cited on pages 34, 35, 43, and 55.

[49] P.J. Lee and E.F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *EUROCRYPT '88, Lecture Notes in Computer Science*, pages 275–280, 1988. Cited on pages 37 and 59.

[50] A. K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 14(4):255–293, 2001. Cited on pages 101, 103, and 105.

[51] J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988. Cited on page 59.

[52] Y. X. Li, R. H. Deng, and X.-M. Wang. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, 40(1):271–, 1994. Cited on page 28.

[53] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes*, volume 16. North-Holland Mathematical Library, 1977. Cited on pages 22 and 90.

[54] A. May, A. Meurer, and E. Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011. Cited on page 60.

[55] R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DNS Progress Report*, pages 114–116, 1978. Cited on pages 19, 24, and 27.

[56] C. Aguilar Melchor, P.-L. Cayrel, and P. Gaborit. A new efficient threshold ring signature scheme based on coding theory. In J. Buchmann and J. Ding, editors, *PQCrypto*, volume 5299 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2008. Cited on page 20.

[57] A. Menezes, M. Qu, D. Stinson, and Y. Wang. Evaluation of security level of cryptography: Esign signature scheme. In *CRYPTREC Project*, 2001. Cited on page 103.

[58] R. C. Merkle. A certified digital signature. In Brassard [19], pages 218–238. Cited on page 31.

[59] D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In M. Charikar, editor, *SODA*, pages 1468–1480. SIAM, 2010. Cited on pages 59 and 94.

[60] L. Minder and A. Sinclair. The extended k-tree algorithm. In *SODA*, pages 586–595, 2009. Cited on pages 19, 37, 38, 61, 65, 79, 80, 81, 84, 85, and 102.

[61] R. Misoczki and P. S. L. M. Barreto. Compact McEliece keys from Goppa codes. In *Selected Areas in Cryptography – SAC'2009*, volume 5867 of *Lecture Notes in Computer Science*, pages 276–392. Springer, 2009. Cited on pages 20, 24, 25, 62, 69, 79, 83, and 85.

[62] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965. Cited on page 106.

[63] R. Niebuhr. *Application of Algebraic-Geometric Codes in Cryptography*. Vdm Verlag Dr. Müller, 2008. Cited on pages 34, 35, and 43.

[64] R. Niebuhr. Critical attacks in code-based cryptography. In *WEWoRC 2011*, 2011. Cited on page 34.

[65] R. Niebuhr and P.-L. Cayrel. Broadcast attacks against code-based encryption schemes. Lecture Notes in Computer Science. Springer, 2011. Accepted for WEWoRC 2011 post-proceedings. Cited on page 35.

[66] R. Niebuhr, P.-L. Cayrel, and J. Buchmann. Improving the efficiency of Generalized Birthday Attacks against certain structured cryptosystems. In *WCC 2011*, pages 163–172. Springer, Apr 2011. Cited on pages 58 and 79.

[67] R. Niebuhr, P.-L. Cayrel, S. Bulygin, and J. Buchmann. On lower bounds for Information Set Decoding over $\mathbb{F}_q$. In *SCC 2010, RHUL, London, UK*, pages 143–157, 2010. Cited on pages 65, 85, 92, 105, 106, and 109.

[68] R. Niebuhr, P.-L. Cayrel, S. Bulygin, and J. Buchmann. On lower bounds for Information Set Decoding over $\mathbb{F}_q$ and on the effect of Partial Knowledge. Submitted to Math in CS (SCC 2010), 2011. Cited on pages 92 and 105.

[69] R. Niebuhr, M. Meziani, S. Bulygin, and J. Buchmann. Selecting Parameters for Secure McEliece-based Cryptosystems. *International Journal of Information Security*, 2011. Cited on pages 37 and 109.

[70] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986. Cited on pages 19 and 27.

[71] C. O'Eigeartaigh and M. Scott. Pairing calculation on supersingular genus 2 curves. In *Selected Areas in Cryptography*, pages 302–316, 2006. Cited on page 103.

[72] R. Overbeck. Statistical decoding revisited. In L. M. Batten and R. Safavi-Naini, editors, *ACISP*, volume 4058 of *Lecture Notes in Computer Science*, pages 283–294. Springer, 2006. Cited on pages 59, 61, 85, and 90.

[73] Y. Pan and Y. Deng. A broadcast attack against NTRU using Ding's algorithm. Cryptology ePrint Archive, Report 2010/598, 2010. `http://eprint.iacr.org/`. Cited on page 45.

[74] C. Peters. Information-set decoding for linear codes over $\mathbb{F}_q$. In *PQCrypto*, pages 81–94, 2010. Cited on pages 60, 68, 70, and 92.

[75] C. Peters. Iteration and operation count for information-set decoding over $\mathbb{F}_q$, Jan 2010. `http://www.win.tue.nl/~cpeters/isdfq.html`. Cited on pages 58, 68, 69, and 70.

[76] E. Petrank and R. M. Roth. Is code equivalence easy to decide? *IEEE Transactions on Information Theory*, 43:1602–1604, 1997. Cited on page 39.

[77] T. Plantard and W. Susilo. Broadcast attacks against lattice-based cryptosystems. In *Proceedings of the 7th International Conference on Applied Cryptography and Network Security*, ACNS '09, pages 456–472, Berlin, Heidelberg, 2009. Springer-Verlag. Cited on page 44.

[78] D. Pointcheval. Chosen-ciphertext security for any one-way cryptosystem. In *Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, pages 129–146, London, UK, 2000. Springer-Verlag. Cited on page 54.

[79] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, pages 5–9, 1962. Cited on pages 19, 59, 67, and 92.

[80] J. A. Ryan and P. Fitzpatrick. Enumeration of inequivalent irreducible Goppa codes. *Discrete Applied Mathematics*, 154:399–412, February 2006. Cited on page 108.

[81] N. Sendrier. Finding the permutation between equivalent linear codes: the support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2000. Cited on pages 39 and 108.

[82] N. Sendrier. On the security of the McEliece public-key cryptosystem. In M. Blaum, P.G. Farrell, and H. van Tilborg, editors, *Information, Coding and Mathematics*, pages 141–163. Kluwer, 2002. Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday. Cited on pages 48 and 51.

[83] N. Sendrier. Decoding one out of many. In B.-Y. Yang, editor, *PQCrypto*, volume 7071 of *Lecture Notes in Computer Science*, pages 51–67. Springer, 2011. Cited on pages 39 and 113.

[84] N. Sendrier and B. Biswas. HyMES – hybrid McEliece encryption scheme, May 2011. Cited on pages 27, 28, and 46.

[85] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27:379–423, jul 1948. Cited on page 21.

[86] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26:1484–1509, 1997. Cited on page 25.

[87] V.M. Sidelnikov and S.O. Shestakov. On the insecurity of cryptosystems based on generalized Reed-Solomon Codes. *Discrete Mathematics*, 1(4):439–444, 1992. Cited on pages 19, 28, and 34.

[88] J. Stern. A method for finding codewords of small weight. In *Proceedings of Coding Theory and Applications*, pages 106–113, 1989. Cited on pages 37 and 59.

[89] J. Stern. A new identification scheme based on syndrome decoding. In *CRYPTO*, pages 13–21, 1993. Cited on pages 30 and 85.

[90] H. Stichtenoth. *Algebraic Function Fields and Codes*. Springer-Verlag, 1993. Cited on page 24.

[91] F. Strenzke. A smart card implementation of the mceliece pkc. In P. Samarati, M Tunstall, J. Posegga, K. Markantonakis, and D. Sauveron, editors, *Information*

Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices, volume 6033 of *Lecture Notes in Computer Science*, pages 47–59. Springer Berlin / Heidelberg, 2010. Cited on page 102.

[92] H.-M. Sun. Further cryptanalysis of the McEliece public-key cryptosystem. *IEEE Communications Letters*, 4(1):18–19, Jan 2000. Cited on page 35.

[93] G. Szewczyk. The dynamic ciphers: New concept of long-term content protecting. *Annales Universitatis Apulensis Series Oeconomica*, 2(10):34, 2008. Cited on page 103.

[94] V. G. Umana and G. Leander. Practical key recovery attacks on two McEliece variants. In *International Conference on Symbolic Computation and Cryptography – SCC'2010*, 2010. To appear. Cited on pages 62 and 101.

[95] J. van Tilburg. On the McEliece public-key cryptosystem. In *CRYPTO '88, Lecture Notes in Computer Science*, volume 403, pages 119–131. Springer, 1988. Cited on page 59.

[96] R. R. Varshamov. Estimate of the number of signals in error correcting codes. *Dokl. Acad. Nauk SSSR*, 117:739–741, 1957. English translation in [44], pp. 68–71. Cited on page 23.

[97] D. Wagner. A generalized birthday problem. In *CRYPTO*, pages 288–303, 2002. Cited on pages 19, 37, 79, and 81.

[98] D. Zheng, X. Li, and K. Chen. Code-based ring signature scheme. *I. J. Network Security*, 5(2):154–157, 2007. Cited on pages 20 and 102.

Lists of symbols and abbreviations

General

$\mathcal{I}_k$	Set $\{1, 2, \ldots, k\}$
I_n	Identity matrix of size n
$[n, k]$ code	Linear code of length n and dimension k
$[n, k, d]$ code	An $[n, k]$ code with a minimum distance of d
(n, k, t) code	An $[n, k]$ code with an error-correcting capability of t
$r = n - k$	Co-dimension of an $[n, k]$ code
$R = k/n$	Information rate of an $[n, k]$ code
$\mathcal{D}_\mathcal{C}$, $\mathcal{D}_G$, $\mathcal{D}_H$	Decoding algorithm for a code $\mathcal{C}$, or a code generated by G or H, respectively
$\mathrm{le}(v)$	The leading element of a non-zero vector $v \in \mathbb{F}_q^n$, i.e. the first non-zero element
$\mathcal{W}_{n,t,q}$	Set of words over $\mathbb{F}_q$ of length n and weight t
$\mathrm{wt}(v)$	The (Hamming) weight of vector v
$\mathrm{d}(x, y)$	The (Hamming) distance between two vectors x and y

CFS	Courtois-Finiasz-Sendrier
CVE	Cayrel-Véron-El Yousfi
CWE	Constant weight encoding
FSB	Fast Syndrome-Based
GBA	Generalized Birthday Algorithm
GRS	Generalized Reed-Solomon
GV	Gilbert-Varshamov
ID	Identification
ISD	Information-Set Decoding
LV	Lenstra-Verheul
pk, sk	Public key, secret key
PKC	Public-Key Cryptosystem
QC, QD	Quasi-cyclic, quasi-dyadic
SSA	Support Splitting Algorithm

Algorithms

PRG(x)	Cryptographically secure pseudo random number generator from fixed length random seed
l_q	$\lfloor \log_q \binom{n}{t} \rfloor$
$\varphi()$	Bijective function mapping an integer in $\mathbb{Z}_{q^l} = \mathbb{Z}/q^l\mathbb{Z}$ to a word of length n and weight t. We apply φ to vectors in $\mathbb{F}_q^k$ by enumerating these vectors first, and then apply φ
$\mathrm{MSB}_x(v)$	The left x bits of v
$\mathrm{LSB}_x(v)$	The right x bits of v
$\mathrm{len}(v)$	Length of vector v
$h()$	Cryptographic secure hash function to a word of length l

A. Appendix: Proofs of Propositions

A.1. Proof of Proposition 4.1.3

In this section, we prove Proposition 4.1.3 (page 67). We begin with a detailed description of the efficiency improvement we presented in Section 4.1.3.

Efficiency improvement using the field structure of $\mathbb{F}_q$

For the sake of better readability in this section, we introduce a simplified notation for equation 4.1 (page 63):

$$H = \left(\begin{array}{c|c|c} I_{n-k-l} & 0 & H_1 \\ \hline 0 & I_l & H_2 \end{array}\right) = \left(\begin{array}{c|c} I_{n-k-l} & K_1 \\ \hline 0 & K_2 \end{array}\right).$$

The step of Algorithm 8 that can be made more efficient using the field structure of $\mathbb{F}_q$ is the search for a pair (e_1, e_2) such that $e_1 \in \mathcal{W}_{k+l;\lfloor p/2 \rfloor;q;p_1}$, $e_2 \in \mathcal{W}_{k+l;\lceil p/2 \rceil;q;p_1}$ and

$$K_2 e_1^T = s_2^T - K_2 e_2^T,$$

where $\mathcal{W}_{k+l;p;q;p_1}$ is the set of all q-ary words of length $k+l$, weight p, and weight p_1 on the first l bits.

Let W_1', W_2, L_1', and L_2' be defined as in (4.4)-(4.6). First note that for any pair (e_1, e_2) and all non-zero values $y \in \mathbb{F}_q$, we have

$$K_2 e_1^T = s_2^T - K_2 e_2^T \Leftrightarrow (K_2 e_1^T) y^{-1} = (s_2^T - K_2 e_2^T) y^{-1}.$$

Instead of storing $K_2 e_1^T$ and $s_2^T - K_2 e_2^T$, we can store $(K_2 e_1^T)(\mathrm{le}(K_2 e_1^T))^{-1}$ in L_1' and $(s_2^T - K_2 e_2^T)(\mathrm{le}(s_2^T - K_2 e_2^T))^{-1}$ in L_2', respectively. The list L_1', however, would contain every entry exactly $(q-1)$ times, since for every $y \in \mathbb{F}_q \backslash \{0\}$, e_1 and $y e_1$ yield the same entry. Therefore, we can generate the first list by using only vectors e_1 whose first non-zero entry is 1.

To see that there is exactly one collision between L_1' and L_2' for every solution of the problem, let (e_1, e_2) be a pair found by our algorithm. Let $y = \mathrm{le}(K_2 e_1^T)$ and $z = \mathrm{le}(s^T - K_2 e_2^T)$. Then we have

$$(K_2 e_1^T) y^{-1} = (s^T - K_2 e_2^T) z^{-1},$$

and therefore $(e_1 z y^{-1}, e_2)$ is a solution to the problem.

Conversely, let (e_1, e_2) be a solution to the problem, i.e. $K_2 e_1^T + K_2 e_2^T = s_2^T$. We want to show that there exists a collision between L_1' and L_2' which corresponds to this solution. Let $y = \mathrm{le}(K_2 e_1^T)$ and $z = \mathrm{le}(s_2^T - K_2 e_2^T)$. Since $K_2 e_1^T = s_2^T - K_2 e_2^T$, we have

$$(K_2 e_1^T) y^{-1} = (s_2^T - K_2 e_2^T) z^{-1}. \tag{A.1}$$

As we did not limit the set W_2, the right hand side of equation (A.1) is an element of L_2'.
Let $x = \mathrm{le}(e_1)$. The first non-zero entry of $e_1' = e_1 x^{-1}$ is 1, so it was used to calculate a member of L_1'. As $\mathrm{le}(K_2 e_1'^T) = \mathrm{le}(K_2 (e_1 x^{-1})^T) = y x^{-1}$, we have

$$(K_2 e_1'^T)(\mathrm{le}(K_2 e_1'^T))^{-1} = (K_2 (e_1 x^{-1})^T)(y x^{-1})^{-1} = (K_2 e_1^T) y^{-1}.$$

Therefore, the left hand side of equation (A.1) belongs to L_1'.
Since $z = y$, this collision between L_1' and L_2' corresponds to the solution (e_1, e_2).

Obviously, this improvement can only be applied if $p > 0$, i.e. if there actually is a search for collisions. If $p = 0$, which corresponds to the basic ISD algorithm by Prange, we are simply trying to find a permutation which shifts all error positions into the first r positions of s, so the runtime is the inverse of the probability P_0 of this event with $P_0 = \binom{r}{t} / \binom{n}{t}$. For the rest of the appendix we assume $p > 0$.

Cost of the algorithm

In most cases, the value of t will be smaller than the GV bound, and we expect the algorithm to require many iterations. In that case, in one iteration of our main loop, we expect to test a fraction

$$\begin{aligned} \lambda_q(z_q) &= 1 - \left(1 - \frac{1}{\binom{k}{p_1}\binom{l}{p_2}(q-1)^{p-1}}\right)^{|W_1'| \cdot |W_2|} \\ &\approx 1 - \exp\left(-\frac{|W_1'| \cdot |W_2|}{\binom{k}{p_1}\binom{l}{p_2}(q-1)^{p-1}}\right) \\ &= 1 - \exp(-z_q) \end{aligned}$$

of vectors in $\mathcal{W}_{k+l;p;q}$, where

$$z_q = \frac{|W_1'| \cdot |W_2|}{\binom{k}{p_1}\binom{l}{p_2}(q-1)^{p-1}}. \tag{A.2}$$

The approximation via the exponential function is possible since Proposition 4.1.3 requires that z_q is small and hence $\exp(-z_q) \approx 1 - z_q$. This slightly overestimates the success probability, but the difference is less than 2^{-12} for relevant parameters.

The success probability of each pair (e_1, e_2) is the number of error combinations matching the syndrome in the last l rows, divided by the total number of possible values He^T

with $e \in \mathcal{W}_{k+l;p;q}$. Depending on the code parameters, the latter is either given by the number of error patterns or by the number of syndromes:

$$P_q = \frac{\lambda_q(z_q)\binom{r-l}{t-p}(q-1)^{t-p+1}}{\min\left(\binom{n}{t}(q-1)^t, q^r\right)}.$$

The success probability in one iteration of main loop is thus:

$$\begin{aligned} P_{p;q}(l) &= 1-(1-P_q)^{|W_1'|\cdot|W_2|} \\ &\approx 1-\exp(-P_q\cdot|W_1'|\cdot|W_2|) \\ &= 1-\exp\left(-\frac{\lambda_q(z_q)}{N_{p;q}(l)}\right), \end{aligned}$$

where

$$N_{p;q}(l) = \frac{\min\left(\binom{n}{t}(q-1)^t, q^r\right)}{\binom{r-l}{t-p}|W_1'|\cdot|W_2|(q-1)^{t-p+1}}.$$

As described above, the set W_1' (and, similarly, W_2) is computed by storing partial sums, reducing the cost of each calculation to $l\frac{q-1}{q}$ operations. This is done for the rightmost $p_1' = \lceil p_1/2 \rceil$ error positions. Then the $p_2' = \lceil p_2/2 \rceil$ remaining positions are added, again storing partial sums, but at a cost of only one operation each, since the corresponding vectors have only one non-zero entry. If the sets are chosen to maximize this effect, every combination of p_1' error locations allows to compute $(\binom{l}{p_2'}(q-1)^{p_2'})$ vectors at the cost of only one operation. This leads to a total cost of

$$\frac{|W_1'|}{\binom{l}{p_2'}(q-1)^{p_2'}}\left(\frac{(q-1)l}{q}+\binom{l}{p_2'}(q-1)^{p_2'}p_2'\right) = \frac{(q-1)l|W_1'|}{q\binom{l}{p_2'}(q-1)^{p_2'}} + |W_1'|p_2'$$

to compute the syndromes corresponding to the vectors in W_1', and similarly for W_2.

For small parameters, the optimal p_2 might be odd, and the sets W_1' and W_2 will have different size. However, the difference between the optimal runtimes of an odd p_2 and p_2+1 is very small, thus we will assume an even p_2.

For small $P_{p;q}(l)$, the cost of the algorithm can be approximated to

$$\frac{N_{p;q}(l)}{\lambda_q(z_q)}\cdot\left(\frac{(q-1)l|W_1'|}{q\binom{l}{p_2'}(q-1)^{p_2'}} + |W_1'|p_2' + \frac{(q-1)l|W_2|}{q\binom{l}{p_2'}(q-1)^{p_2'}} + |W_2|p_2' + K_q\frac{\lambda_q(z_q)|W_1'|\cdot|W_2|}{q^l}\right),$$

which is the approximate number of iterations times the number of operations per iteration. K_q is the expected cost to perform the check that $\mathrm{wt}(s^T - H(e_1+e_2)^T) = t-p$.

It is easy to see that choosing $|W_1'| = |W_2|$ minimizes this expression.

$$\begin{aligned} & N_{p;q}(l) \cdot \left(\frac{|W_1'|}{\lambda_q(z_q)} \left(\frac{2(q-1)l}{q\binom{l}{p_2'}(q-1)^{p_2'}} + p_2 \right) + K_q \frac{|W_1'|^2}{q^l} \right) \\ = \; & \frac{\min\left(\binom{n}{t}(q-1)^t, q^r\right)}{\binom{r-l}{t-p}|W_1'|^2(q-1)^{t-p}} \cdot \left(\frac{|W_1'|}{\lambda_q(z_q)} \left(\frac{2(q-1)l}{q\binom{l}{p_2'}(q-1)^{p_2'}} + p_2 \right) + K_q \frac{|W_1'|^2}{q^l} \right). \end{aligned}$$

This expression decreases as the size of W_1' increases, so we maximize this size, choose $z_q = 1$ and set $\lambda_q = \lambda_q(1) = 1 - e^{-1}$. Using (A.2), we get

$$N_{p;q}(l) \cdot \left(\lambda_q^{-1} \left(\frac{2(q-1)l}{q\binom{l}{p_2'}(q-1)^{p_2'}} + p_2 \right) \sqrt{\binom{k}{p_1}\binom{l}{p_2}(q-1)^{p-1}} + K_q \frac{\binom{k}{p_1}\binom{l}{p_2}(q-1)^{p-1}}{q^l} \right),$$

where $N_{p;q}$ is now

$$N_{p;q}(l) = \frac{\min\left(\binom{n}{t}(q-1)^t, q^r\right)}{\binom{r-l}{t-p}\binom{k}{p_1}\binom{l}{p_2}(q-1)^t}.$$

Minimizing over p_1, p_2 and l gives the result.

Now consider the case where $\binom{r}{t-p}\binom{k}{p}(q-1)^t \geq q^r$. Then the main loop is likely to succeed after a single iteration. This corresponds to the birthday algorithm described in [36]:

$$\mathrm{WF}_{\mathrm{BA}} \approx \frac{2}{\sqrt{P}} \cdot \left(l + \frac{K_0}{2\sqrt{P}2^l} \right).$$

We can apply this result here, since it does not depend on the field size, but only on the success probability. In the q-ary case, this expression becomes

$$\mathrm{WF}_{\mathrm{qBA}} \approx \frac{2}{\sqrt{P}} \cdot \left(l + \frac{K_0}{2\sqrt{P}q^l} \right).$$

Easy analysis shows that the optimal value for l is

$$l = \log_q \left(\frac{\ln(q) K_0}{2\sqrt{P}} \right).$$

Applying this in our case with K_q instead of K_0 (since K_0 is the cost of the third step in the algorithm of [36], which corresponds to K_q when applied in the case of ISD), using

$$P = P_q \approx \frac{\binom{r-l}{t-p}(q-1)^{t-p+1}}{q^r},$$

and minimizing over p and l yields the lower bound result:

$$\mathrm{WF}_{\mathrm{qISD}}(n,r,t,q) \approx \frac{2lq^{r/2}}{\sqrt{\binom{r-l}{t-p}(q-1)^{t-p+1}}}.$$

A.2. Proof of Proposition 4.2.1

The proof follows the same approach as above, and we focus on the differences compared to that case.

Since the domain of the error values has been restricted from $\mathcal{W}_{n;p;q} \in \mathbb{F}_q^n$ to $\mathcal{W}_{n;p;q} \cap E^n$, every iteration of the algorithm tests a larger ratio of possible error vectors. We continue to denote this ratio by $\lambda_q(z_q) = 1 - \exp(z_q)$, where z_q changes and is now defined as

$$z_q = \frac{|W_1'| \cdot |W_2|}{\binom{k}{p_1}\binom{l}{p_2}|E|^{p-1}}. \tag{A.3}$$

This also changes the success probability P_q of each pair (e_1, e_2) to

$$P_q = \frac{\lambda_q(z_q)\binom{r-l}{t-p}|E|^{t-p+1}}{\min\left(\binom{n}{t}|E|^t, q^r\right)}.$$

Therefore, we get a success probability of each iteration of the main loop of

$$\begin{aligned} P_{p;q}(l) &= 1 - (1 - P_q)^{\binom{k}{p_1}\binom{l}{p_2}|E|^p} \\ &\approx 1 - \exp\left(-P_q \cdot \binom{k}{p_1}\binom{l}{p_2}|E|^p\right) \\ &= 1 - \exp\left(-\frac{\lambda_q(z_q)}{N_{p;q}(l)}\right), \end{aligned}$$

with

$$N_{p;q}(l) = \frac{\min\left(\binom{n}{t}|E|^t, q^r\right)}{\binom{r-l}{t-p}|W_1'| \cdot |W_2| \cdot |E|^t}.$$

Hence, for small $P_{p;q}(l)$, the cost of the algorithm is

$$\frac{\min\left(\binom{n}{t}(q-1)^t, q^r\right)}{\binom{r-l}{t-p}|W_1'|^2(q-1)^{t-p}} \cdot \left(\frac{|W_1'|}{\lambda_q(z_q)}\left(\frac{2|E|l}{(|E|+1)\binom{l}{p_2'}(q-1)^{p_2'}} + p_2\right) + K_q\frac{|W_1'|^2}{q^l}\right).$$

Using (A.2) and setting $z_q = 1$, we get

$$N_{p;q}(l) \cdot \left(\lambda_q^{-1}\left(\frac{2|E|l}{(|E|+1)\binom{l}{p_2'}(q-1)^{p_2'}} + p_2\right)\sqrt{\binom{k}{p_1}\binom{l}{p_2}|E|^{p-1}} + K_q\frac{\binom{k}{p_1}\binom{l}{p_2}|E|^{p-1}}{q^l}\right),$$

where $N_{p;q}$ is now

$$N_{p;q}(l) = \frac{\min\left(\binom{n}{t}|E|^t, q^r\right)}{\binom{r-l}{t-p}\binom{k}{p_1}\binom{l}{p_2}|E|^t}.$$

Minimizing over p_1, p_2 and l gives the result.

Similarly, in the case where $\binom{r}{t-p}\binom{k}{p}|E|^t \geq q^r$, we use the formula

$$\mathrm{WF}_{\mathrm{qBA}} \approx \frac{2}{\sqrt{P}} \cdot \left(l + \frac{K_0}{2\sqrt{P}q^l} \right)$$

with

$$P = P_q \approx \frac{\binom{r-l}{t-p}|E|^{t-p+1}}{q^r}.$$

Minimizing over p and l yields the lower bound result:

$$\mathrm{WF}_{\mathrm{qISD}}(n,r,t,q) \approx \frac{2lq^{r/2}}{\sqrt{\binom{r-l}{t-p}|E|^{t-p+1}}}.$$

A.3. Proof of Proposition 4.2.2

We mention only the changes compared with the proof in Appendix A.2.

Since the size of the sets W_i changes, the values of z_q, P_q and $N_{p;q}(l)$ become

$$z_q = \frac{|W_1| \cdot |W_2|}{\binom{k}{p_1}\binom{l}{p_2}\binom{t}{p}p!}, \tag{A.4}$$

$$P_q = \frac{\lambda_q \binom{r-l}{t-p}(t-p)!}{\min(\binom{n}{t}t!, q^r)},$$

$$N_{p;q}(l) = \frac{\min(\binom{n}{t}t!, q^r)}{\binom{r-l}{t-p}\binom{k}{p_1}\binom{l}{p_2}\binom{t}{p}p!(t-p)!}.$$

We can use the same strategy to efficiently compute the syndromes. In this case, the total cost is

$$\frac{|W_1'|}{\binom{l}{p_2'}\binom{t}{p_2'}p_2'!}\left(\frac{(q-1)l}{q} + \binom{l}{p_2'}\binom{t}{p_2'}p_2'! \cdot p_2'\right) = \frac{(q-1)l|W_1'|}{q\binom{l}{p_2'}\binom{t}{p_2'}p_2'!} + |W_1'|p_2'$$

to compute the syndromes corresponding to the vectors in W_1', and similarly for W_2.

Using (A.4) and following the same steps as above, we get

$$N_{p;q}(l) \cdot \left(\lambda_q^{-1} \left(\frac{2(q-1)l}{q\binom{l}{p_2'}\binom{t}{p_2'}p_2'!} + p_2 \right) \sqrt{\binom{k}{p_1}\binom{l}{p_2}\binom{t}{p}p!} + K_q \frac{\binom{k}{p_1}\binom{l}{p_2}\binom{t}{p}p!}{q^l} \right),$$

Minimizing over l, p_1 and p_2 gives the result.

In the second case, we can use the same formula as above with the probability changed, i.e.

$$\mathrm{WF}_{\mathrm{qBA}} \approx \frac{2}{\sqrt{P}} \cdot \left(l + \frac{K_0}{2\sqrt{P} q^l} \right).$$

With

$$l = \log_q \left(\frac{\ln(q) K_0}{2\sqrt{P}} \right),$$

$$P = P_q \approx \frac{\binom{r-l}{t-p}(t-p)!}{q^r}.$$

and minimizing over p and l yields the lower bound result:

$$\mathrm{WF}_{\mathrm{qBA}}(n, r, t, q) \approx \frac{2 l q^{r/2}}{\sqrt{\binom{r-l}{t-p}(t-p)!}}.$$

Minimizing over p yields the result as above.

B. Appendix: Implementations

Broadcast attack

Listing B.1: Broadcast attack

```
1  import java.io.*;
2  import java.security.KeyPair;
3  import java.security.KeyPairGenerator;
4  import java.security.PrivateKey;
5  import java.security.PublicKey;
6  import java.security.SecureRandom;
7  import java.security.Security;
8  import java.util.Calendar;
9
10 import javax.crypto.Cipher;
11 import javax.crypto.CipherInputStream;
12 import javax.crypto.CipherOutputStream;
13
14 import de.flexiprovider.core.FlexiCoreProvider;
15 import de.flexiprovider.pqc.*;
16 import de.flexiprovider.pqc.ecc.*;
17 import de.flexiprovider.pqc.ecc.niederreiter.*;
18 import de.flexiprovider.common.math.linearalgebra.*;
19
20 import Jama.*;
21
22 public class Broadcast {
23
24     //Constants
25     final static int recipients=2;
26
27     //Variables
28     static KeyPairGenerator kpg;
29     static ECCKeyGenParameterSpec eccParams;
30     static KeyPair[] keyPair;
31     static PublicKey[] pubKey;
32     static PrivateKey[] privKey;
33     static String message;
34     static byte[] messageBytes;
35     static byte[][] ciphertextBytes;
```

```
36      static Jama.Matrix[] jm;
37      static Jama.Matrix[] syn;
38      static Jama.Matrix mes;
39
40
41      public static void main(String[] args) throws Exception {
42
43          initialize();
44
45          //Generate KeyPairs
46          genKeyPairs();
47
48          //Set up a message
49          ciphertextBytes=encrypt("test", pubKey, privKey);
50          byte[] pad = padMessage(messageBytes);
51
52          //Encode message using each public key
53          GF2Vector m = Conversions.encode(
54              ((NiederreiterPublicKey)pubKey[0]).getN(),
55              ((NiederreiterPublicKey)pubKey[0]).getT(), pad);
56              Vector hm = ((NiederreiterPublicKey)pubKey[0]).getH
                    ().rightMultiplyRightCompactForm(m);
57
58              //Set up vectors and matrices for the linear
                    equation system
59              int numRows=((NiederreiterPublicKey)pubKey[0]).getK
                    ();
60              int numCols=((NiederreiterPublicKey)pubKey[0]).getN
                    ();
61
62              GF2Matrix[] gfM = new GF2Matrix[recipients];
63              jm = new Jama.Matrix[recipients];
64              syn = new Jama.Matrix[recipients];
65              mes = new Jama.Matrix(numCols, 1);
66              for(int i=0;i<numCols;i++)mes.set(i, 0, m.getBit(i))
                    ;
67
68              //Assign public keys to corresponding matrices
69              //and ciphertexts to syndrome vectors
70              for(int i=0;i<recipients;i++){
71                  gfM[i] = ((NiederreiterPublicKey)pubKey[i]).getH
                        ().extendRightCompactForm();
72                  jm[i] = M2JM(gfM[i]);
73                  syn[i] = jm[i].times(mes);
74              }
```

```
75
76          //Concatenate matrices and vectors
77          Jama.Matrix jjm=verticalJoin(jm);
78          Jama.Matrix jsyn=verticalJoin(syn);
79          normalize(jsyn, 2);
80
81          //Solve equation system
82          long tim=Calendar.getInstance().getTimeInMillis();
83          Jama.Matrix solution=modSolve(jjm, jsyn);
84          System.out.println((Calendar.getInstance().
                getTimeInMillis()-tim)/1000+" Seconds");
85          normalize(solution, 2);
86
87          //Print solution
88          GF2Vector lsgVec=vec2GFvec(solution);
89          byte[] lsgByte=Conversions.decode(
90               ((NiederreiterPublicKey)pubKey[0]).getN(),
91               ((NiederreiterPublicKey)pubKey[0]).getT(),lsgVec);
92          System.out.println("Padded message");
93          for(int i=0;i<pad.length;i++)System.out.print(pad[i]+" "
                );
94          System.out.println("Padded solution");
95          for(int i=0;i<lsgByte.length;i++)
96               System.out.print(lsgByte[i]+" ");
97          String s=new String(unPadMessage(lsgByte));
98          System.out.println("Cleartext again");
99          System.out.println(s);
100
101         //Check solution
102         if(solution==null)
103              System.out.println("Solution is null");
104         if(solution!=null&&JMeq(mes,solution))
105              System.out.println("");
106         else {
107              //If wrong solution, output debug info
108              System.out.println(":(");
109              for(int i=0;i<mes.getRowDimension();i++)
110                   System.out.print(mes.get(i,0)+" ");
111              System.out.println();
112              for(int i=0;i<solution.getRowDimension();i++)
113                   System.out.print(solution.get(i,0)+" ");
114              System.out.println();
115              Jama.Matrix check = jjm.times(solution);
116              normalize(check, 2);
117              if(JMeq(check,jsyn))
```

```
118                 System.out.println("Check correct");
119             for(int i=0;i<jsyn.getRowDimension();i++)
120                 System.out.print(jsyn.get(i,0)+" ");
121             System.out.println();
122             for(int i=0;i<check.getRowDimension();i++)
123                 System.out.print(check.get(i,0)+" ");
124         }
125     }
126
127     static GF2Vector vec2GFvec(Jama.Matrix v){
128         //Assuming v is a column vector
129         GF2Vector ret=new GF2Vector(v.getRowDimension());
130         for(int i=0;i<ret.getLength();i++)
131         if(v.get(i, 0)==1)ret.setBit(i);
132
133         return ret;
134     }
135
136     static Jama.Matrix modSolve(Jama.Matrix M, Jama.Matrix v){
137         //Require m>=n
138         int numCol=M.getColumnDimension();
139         int numRow=M.getRowDimension();
140         System.out.println("ModSolve activated. Parameters "+
                numCol+" columns and "+numRow+" rows.");
141         if(numRow<numCol){
142             System.out.println("Number of rows too small");
143             return null;
144         }
145
146         Jama.Matrix A=M.copy();
147         Jama.Matrix b=v.copy();
148         if(numRow!=b.getRowDimension())return null;
149
150         Jama.Matrix ret=new Jama.Matrix(numCol,1);
151
152         //Initialize permutation tracker
153         int[] perm = new int[numCol];
154         for(int i=0;i<perm.length;i++)perm[i]=i;
155
156         for(int col=0;col<numCol;col++){
157             if(A.get(col, col)==0){
158                 //Need to swap columns
159                 int pos=col;
160                 while(A.get(col, pos)==0&&pos<numCol-1)pos++;
161                 if(pos>=numCol||A.get(col, pos)==0){
```

```
162                     //No better columns found. Trying rows
163                     pos=col;
164                     while(A.get(pos, col)==0&&pos<numRow-1)
165                         pos++;
166 
167                     if(pos>=numRow||A.get(pos, col)==0){
168                         //No row found
169                         System.out.println("Matrix is rank
                                deficient");
170                         A.print(2,0);
171                         b.print(2,0);
172                         return null;
173                     }else{
174                         //Row Pos found. Swapping
175                         swapRows(A, pos, col);
176                         swapRows(b, pos, col);
177                     }
178                 }else{
179                     //Col Pos found, swapping
180                     swapCols(A, col, pos);
181                     int temp=perm[col]; perm[col]=perm[pos];
                            perm[pos]=temp;
182                 }
183             }
184 
185             //Pivot is 1 now
186             for(int row=0;row<numRow;row++)
187                 if(row!=col&&A.get(row, col)==1){
188                     addRows(A, row, col);
189                     addRows(b, row, col);
190                 }
191         }
192 
193         //Done diagonalizing
194         for(int i=0;i<ret.getRowDimension();i++)
195             ret.set(perm[i], 0, b.get(i, 0));
196         return ret;
197     }
198 
199     static Jama.Matrix applyPerm(Jama.Matrix v, int[] p){
200         if(v.getRowDimension()!=p.length){
201             System.out.println("Permutation dimensions don't
                    match");
202             return null;
203         }
```

```
204         Jama.Matrix ret=new Jama.Matrix(v.getRowDimension(),1);
205         for(int i=0;i<p.length;i++)
206             ret.set(p[i], 0, v.get(i, 0));
207         return ret;
208     }
209
210     static void addRows(Jama.Matrix A, int a, int b){
211         //Adds a+b and stores it as row a
212         Jama.Matrix temp =
213             A.getMatrix(a,a,0,A.getColumnDimension()-1).plus(
214             A.getMatrix(b,b,0,A.getColumnDimension()-1));
215         normalize(temp, 2);
216         A.setMatrix(a, a, 0, A.getColumnDimension()-1, temp);
217     }
218
219     static void swapRows(Jama.Matrix A, int a, int b){
220         Jama.Matrix temp =
221             A.getMatrix(a, a, 0, A.getColumnDimension()-1);
222         A.setMatrix(a, a, 0, A.getColumnDimension()-1, A.
                getMatrix(b, b, 0, A.getColumnDimension()-1));
223         A.setMatrix(b, b, 0, A.getColumnDimension()-1, temp);
224     }
225
226     static void swapCols(Jama.Matrix A, int a, int b){
227         Jama.Matrix temp =
228             A.getMatrix(0, A.getRowDimension()-1, a, a);
229         A.setMatrix(0, A.getRowDimension()-1, a, a, A.getMatrix
                (0, A.getRowDimension()-1, b, b));
230         A.setMatrix(0, A.getRowDimension()-1, b, b, temp);
231     }
232
233     static void normalize(Jama.Matrix A, int q){
234         int temp;
235         for(int i=0;i<A.getRowDimension();i++)
236             for(int j=0;j<A.getColumnDimension();j++){
237                 temp=(int)(A.get(i, j)%q);
238                 if(temp<0)temp*=-1;
239                 A.set(i, j, temp);
240             }
241     }
242
243     static boolean JMeq(Jama.Matrix A, Jama.Matrix B){
244         if(A.getColumnDimension()!=B.getColumnDimension())
245             return false;
246         if(A.getRowDimension()!=B.getRowDimension())
```

```
247             return false;
248         for(int i=0;i<A.getRowDimension();i++)
249             for(int j=0;j<A.getColumnDimension();j++)
250                 if(A.get(i,j)!=B.get(i, j))return false;
251         return true;
252     }
253 
254     static Jama.Matrix M2JM(GF2Matrix gm){
255         int[][] gma = gm.getIntArray();
256         int numColumns = gm.getNumColumns();
257         int numRows = gm.getNumRows();
258         Jama.Matrix ret = new Jama.Matrix(numRows, numColumns);
259         String bin;
260         int offset;
261         //Set matrix
262         for(int i=0;i<ret.getRowDimension();i++){
263             for(int j=0;j<gma[0].length;j++){
264                 int val=gma[i][j];
265                 bin=Integer.toBinaryString(val);
266                 offset=32-bin.length();
267                 for(int bits=offset;bits<32;bits++){
268                     ret.set(i, j*32+31-bits, Integer.parseInt(
                            bin.substring(bits-offset, bits+1-offset)
                            ));
269                 }
270             }
271         }
272         return ret;
273     }
274 
275     static byte[] unPadMessage(byte[] input) {
276         byte[] result = new byte[input.length - 1];
277         System.arraycopy(input, 1, result, 0, input.length-1);
278         return result;
279     }
280 
281 
282     static byte[] padMessage(byte[] input) {
283         byte[] result = new byte[input.length + 1];
284         result[0] = 0x01;
285         System.arraycopy(input, 0, result, 1, input.length);
286         return result;
287     }
288 
289     static Jama.Matrix randomModularMatrix(int m, int n, int q){
```

```
290         Jama.Matrix ret=new Jama.Matrix(m, n);
291         for(int i=0;i<m;i++)
292             for(int j=0;j<n;j++)
293                 ret.set(i,j,(int)(Math.random()*q));
294         return ret;
295     }
296
297     static Jama.Matrix horizontalJoin(Jama.Matrix A, Jama.Matrix
        B){
298         if(A.getRowDimension()!=B.getRowDimension())
299             return null;
300         Jama.Matrix ret=new Jama.Matrix(A.getRowDimension(),
301                 A.getColumnDimension()+B.getColumnDimension());
302         ret.setMatrix(0,0,A.getRowDimension()-1,
303                 A.getColumnDimension()-1,A);
304         ret.setMatrix(0,B.getRowDimension()-1,
305                 A.getColumnDimension(),
306                 ret.getColumnDimension()-1,B);
307         return ret;
308     }
309
310     static Jama.Matrix verticalJoin(Jama.Matrix[] matrizen){
311         int size=0;
312         int width=matrizen[0].getColumnDimension();
313         for(int i=0;i<matrizen.length;i++){
314             size+=matrizen[i].getRowDimension();
315             if(matrizen[i].getColumnDimension()!=width)
316                 return null;
317         }
318         Jama.Matrix ret=new Jama.Matrix(size, width);
319         int loc=0;
320         for(int i=0;i<matrizen.length;i++){
321             ret.setMatrix(loc, loc+matrizen[i].getRowDimension()
                    -1, 0, width-1, matrizen[i]);
322             loc+=matrizen[i].getRowDimension();
323         }
324         return ret;
325     }
326
327     static Jama.Matrix verticalJoin(Jama.Matrix A, Jama.Matrix B
        ){
328         if(A.getColumnDimension()!=B.getColumnDimension())
                return null;
329         Jama.Matrix ret=new Jama.Matrix(A.getRowDimension()+
330                 B.getRowDimension(), A.getColumnDimension());
```

```
331         ret.setMatrix(0,A.getRowDimension()-1,0,
332                 A.getColumnDimension()-1,A);
333         ret.setMatrix(A.getRowDimension(),
334                 ret.getRowDimension()-1,0,
335                 B.getColumnDimension()-1,B);
336         return ret;
337     }
338
339     static void initialize(){
340         Security.addProvider(new FlexiCoreProvider());
341         Security.addProvider(new FlexiPQCProvider());
342
343         try{
344             kpg = KeyPairGenerator.getInstance("Niederreiter", "
                    FlexiPQC");
345             eccParams =
346                 new ECCKeyGenParameterSpec(11,50);
347
348             kpg.initialize(eccParams, new SecureRandom());
349         }catch(Exception e){System.out.println(e);}
350     }
351
352     static void genKeyPairs(){
353         keyPair = new KeyPair[recipients];
354         pubKey = new PublicKey[recipients];
355         privKey = new PrivateKey[recipients];
356         for(int i=0;i<recipients;i++){
357             keyPair[i] = kpg.generateKeyPair();
358             pubKey[i] = keyPair[i].getPublic();
359             privKey[i] = keyPair[i].getPrivate();
360         }
361     }
362
363     static void testMatrixFunctions(){
364     /*  Jama.Matrix[] jm = new Jama.Matrix[recipients];
365         Jama.Matrix[] syn = new Jama.Matrix[recipients];
366         Jama.Matrix mes = randomModularMatrix(8,1,2);
367
368         for(int i=0;i<recipients;i++){
369             jm[i] = randomModularMatrix(4,8,2);
370             syn[i] = jm[i].times(mes);
371         }
372
373         Jama.Matrix jjm = verticalJoin(jm);
374         Jama.Matrix jsyn = verticalJoin(syn);
```

```
375             normalize(jsyn, 2);
376
377             Jama.Matrix solution = jjm.solve(jsyn);
378             Jama.Matrix check=jjm.times(solution);
379             mes.print(2,0);
380             jjm.print(2,0);
381             jsyn.print(2,1);
382             solution.print(2, 0);
383             check.print(2, 1);
384         */
385         /*  Jama.Matrix jm = randomModularMatrix(8,4,2);
386             Jama.Matrix mes = randomModularMatrix(4,1,2);
387             Jama.Matrix syn = jm.times(mes);
388             normalize(syn, 2);
389             jm.print(2,0);
390             mes.print(2,0);
391             syn.print(2,0);
392             Jama.Matrix sol=modSolve(jm, syn);
393             sol.print(2, 0);
394         */
395         }
396
397         static byte[][] encrypt(String s, PublicKey[] pk, PrivateKey
                [] sk){
398             message = s;
399             messageBytes = message.getBytes();
400             ciphertextBytes=new byte[pk.length][100];
401             try{
402                 Cipher cipher = Cipher.getInstance("Niederreiter", "
                        FlexiPQC");
403
404                 for(int i=0;i<pk.length;i++){
405                     cipher.init(Cipher.ENCRYPT_MODE, pk[i],
406                             new SecureRandom());
407                     ciphertextBytes[i] =
408                         cipher.doFinal(messageBytes);
409                 }
410                 return ciphertextBytes;
411             }catch(Exception e){System.out.println(e+" in method
                    encrypt");}
412
413             return null;
414         }
415 }
```